AN ARITHMETIC-BASED DETERMINISTIC CENTROID INITIALIZATION METHOD FOR THE k-MEANS CLUSTERING ALGORITHM

Matthew Michael Mayo

Columbus State University

D. Abbott Turner College of Business and Computer Science

The Graduate Program in Applied Computer Science

An Arithmetic-based Deterministic Centroid Initialization Method for the *k*-Means Clustering Algorithm

A Thesis in

Applied Computer Science

by

Matthew Michael Mayo

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2016

Abstract

One of the greatest challenges in k-means clustering is positioning the initial cluster centers, or centroids, as close to optimal as possible, and doing so in an amount of time deemed reasonable. Traditional k-means utilizes a randomization process for initializing these centroids, and poor initialization can lead to increased numbers of required clustering iterations to reach convergence, and a greater overall runtime. This research proposes a simple, arithmetic-based deterministic centroid initialization method which is much faster than randomized initialization. Preliminary experiments suggest that this collection of methods, referred to herein as the sharding centroid initialization algorithm family, often outperforms random initialization in terms of the required number of iterations for convergence and overall time-related metrics and is competitive or better in terms of the reported mean sum of squared errors (SSE) metric. Surprisingly, the sharding algorithms often manage to report more advantageous mean SSE values in the instances where their performance is slower than random initialization.

Contents

Abstract iii

List of Figures viii

List of Tables ix

List of Algorithms x

List of Source Code xi

1 Introduction 1

- 1.1 Problem Definition 4
- 1.2 Proposed Method 6
- 1.3 Expected Results 7
- 1.4 Thesis Outline 8

2 Previous Work 10

- 2.1 Improvements to Random Initialization 11
- 2.2 Alternatives to Random Initialization 12

3 Methodology **15**

3.1 The k-means Clustering Algorithm 15

3.2 Random Centroid Initialization 18

3.3 Sharding Centroid Initialization 21

3.3.1 Naive Sharding 21

3.3.2 Mean Sharding 24

3.3.3 Median Sharding 25

3.4 Measuring Success 26

4 Implementation **28**

4.1 Python 29

4.2 Implementation Overview 31

4.3 k-means Clustering Algorithm 33

4.4 Random Centroid Initialization 35

4.5 Sharding Centroid Initialization 36

4.5.1 Naive Sharding 37

4.5.2 Mean Sharding 40

4.5.3 Median Sharding 40

4.6 Utility Modules 41

4.7 Testing Apparatus 41

5 Results and Evaluation **43**

5.1 Experiment Overview 43

5.2 Datasets 46

5.2.1 Research-specific Artificial Datasets 47

5.2.2 Datasets of Known Clusters from the Literature 54

5.2.3 Datasets of Unknown Clusters from Literature 64

6 Conclusions **70**

6.1 Summary of Research 70

6.2 Future Work 74

A Computer Specifications **76**

B Artificial Dataset Names **78**

Bibliography **79**

List of Figures

1.1 A proposed classical machine learning algorithm taxonomy. . . 3

3.1 Sorting by composite value and sharding. 22
3.2 Shard attribute means become centroid attribute values. . . . 23

4.1 Python popularity among the analytics community. 30
4.2 Project tree structure. 32
4.3 *kmeans.py* module overview. 37

5.1 Dessewffy dataset. 48
5.2 Dessewffy naive sharding versus random initialization, $k = 3$. . 49
5.3 Dessewffy naive sharding versus random initialization, $k = 7$. . 50
5.4 Jelonek dataset. 52
5.5 Jelonek naive sharding versus random initialization, $k = 3$. . . 52
5.6 Jelonek naive sharding versus random initialization, $k = 7$. . . 53
5.7 Ruspini random centroid initialization, $k = 4$. 55
5.8 Ruspini naive sharding centroid initialization, $k = 4$. 56
5.9 Ruspini mean sharding centroid initialization, $k = 4$. 57
5.10 Ruspini median sharding centroid initialization, $k = 4$. 58

5.11 Iris parallel coordinates visualization. 61

6.1 Number of iterations of datasets with known number of clusters. 72

6.2 Number of iterations of datasets with class attributes. 73

List of Tables

5.1 Dataset summary. 47
5.2 Dessewffy results. 50
5.3 Jelonek results. 53
5.4 Ruspini results. 59
5.5 Iris results (without class). 61
5.6 Iris results (with class). 61
5.7 Wine results (without class). 63
5.8 Wine results (with class). 63
5.9 Seeds results (without class). 64
5.10 Seeds results (with class). 64
5.11 3D Road Network results. 66
5.12 Power consumption results. 67
5.13 Summarized results of datasets of known clusters. 68
5.14 Summarized results of datasets of unknown clusters. 69

A.1 Experiment computer hardware. 77

List of Algorithms

1 k-means Clustering 17

2 Random Centroid Initialization 19

3 Centroid Updating 20

4 Naive Sharding Centroid Initialization 24

5 Mean Sharding Centroid Initialization 25

6 Median Sharding Centroid Initialization 25

List of Source Code

4.1 *kmeans.py* module excerpt. 33

4.2 *euclidean()* function from *distances.py* module. 35

4.3 *rand_cent()* function from *centroids.py* module. 35

4.4 *naive_sharding()* function from *centroids.py* module. 38

4.5 *_get_mean()* function from *centroids.py* module. 39

4.6 *mean_sharding()* function excerpt from *centroids.py* module. . 40

4.7 *median_sharding()* function excerpt from *centroids.py* module. 40

5.1 Timing functionality excerpt from *kmeans.py* module. 45

Acknowledgements

I would like to thank the following individuals:

My research co-advisor, Dr. J Dana Eckart, whose input and guidance have been immensely helpful, both in this research and beyond, and someone I am lucky to have stumbled upon during my educational pursuits.

My research co-advisor, Dr. Kongmunvattana, who graciously stepped in when it became necessary, and welcomed me as though I had been under his tutelage from the very start.

The remaining thesis committee members, Drs. Hodhod and Ho, both of whom have been understanding and helpful, and whose interest in this research and process I am grateful for.

Dedication

To my parents, Michael and Susan, who molded me early on, and left an impression.

To my children, Azai and Zoey, *mis pulmones*, the absolute best kids that a father could ever ask for.

And most importantly, to my wife Suchdev, *mi corazón*, the most supportive, loving, intelligent, and beautiful person whom I have ever known. I could live a thousand lives and never again be so lucky.

Chapter 1

Introduction

According to Mitchell, *machine learning* is ”concerned with the question of how to construct computer programs that automatically improve with experience” [1]. Machine learning is interdisciplinary in nature, and employs techniques from the fields of computer science, statistics, and artificial intelligence, among others. The main artifacts of machine learning research are algorithms which facilitate this automatic improvement from experience, algorithms which can be applied in such diverse fields as computer vision, artificial intelligence, and data mining [1].

Fayyad, Piatetsky-Shapiro & Smyth define *data mining* as ”the application of specific algorithms for extracting patterns from data” [2]. This demonstrates that, in data mining, the emphasis is on the application of algorithms, as opposed to on the algorithms themselves. We can define the relationship between machine learning and data mining as follows: data mining is a *pro-*

cess, during which machine learning algorithms are utilized as *tools* to extract potentially-valuable patterns held within datasets.

Clustering is a machine learning method used for analyzing data which does not include pre-labeled classes. During the clustering process, data instances are grouped together using the concept of "maximizing the intraclass similarity and minimizing the interclass similarity" [3]. This translates to the clustering algorithm identifying and grouping instances which are similar to one another, in contrast to ungrouped instances which are less-similar to one another. As clustering does not require the pre-labeling of classes, which is, in contrast, required for classification, it is considered a form of *unsupervised learning* [3], intimating that clustering learns by observation as opposed to learning by example.

k-means is perhaps the most well-known example of a clustering algorithm [4, 5]. As such, it is often the default clustering method in data mining applications, allowing for the unsupervised exploration and discovery of groups of similar items. The k-means clustering algorithm continues to be one of the most popular machine algorithms algorithms used in data mining, and was included in the seminal *Top 10 Algorithms in Data Mining* paper by Wu, et al. [5], which outlines the most influential algorithms based on 3 separate data mining conference attendee surveys.

In the context of data mining, clustering is often used as both a standalone tool for gaining insight into a particular set of data and its distribution, as well as a pre-processing step for further algorithms, such as classification [3]. In the

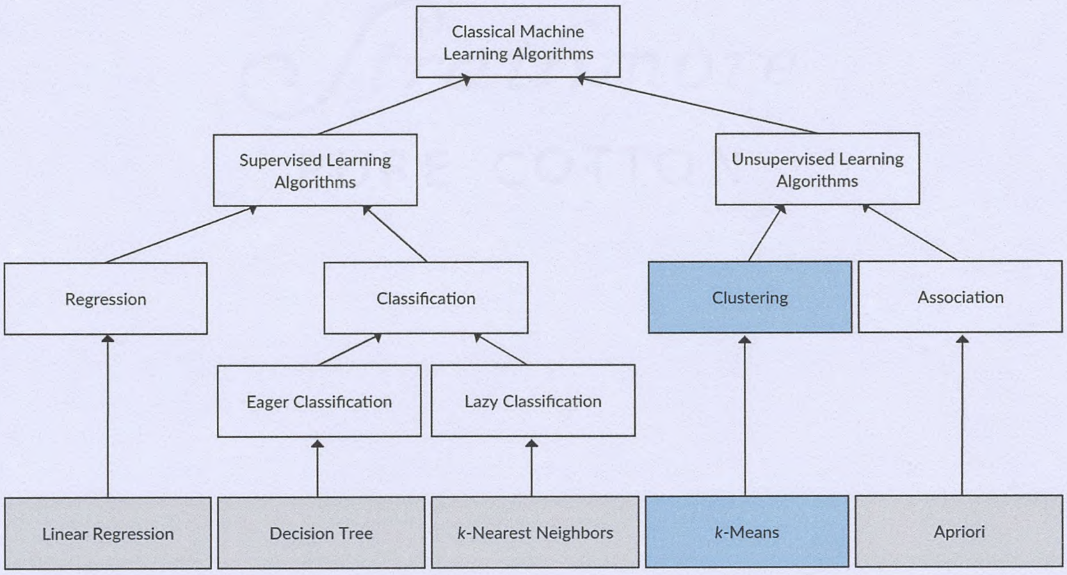


Figure 1.1: A proposed classical machine learning algorithm taxonomy.

former, clustering is the protagonist which enables *pattern recognition* [4], one of the core and original aims of data mining; in the latter, clustering becomes a potentially-vital piece of machine learning pipelines, chains of algorithms which operate serially on a dataset to generate an insightful final outcome. As a member of the *unsupervised learning* branch of classical machine learning algorithms, clustering can be considered a fundamental machine learning technique.

Figure 1.1 presents a proposed classical machine learning algorithm taxonomy - *classical* in the sense that it does not include the more advanced, hybrid, and unorthodox contemporary algorithms - highlighting clustering and the k-means algorithm in blue, with the various leaves representing particular algorithm exemplars.

k-means clustering is an unmistakably simple algorithm; it works by initializing a specified number of generally user-supplied cluster centers, or centroids, and computing the distance, usually Euclidean, between the coordinates of each of a given dataset's instances and the coordinates of each of the centroids [4]. Each instance is added as a member of the cluster of the closest centroid, after which the centroids are recentered to be in the position which minimizes the sum of squared errors (SSE), which is the sum of the squared differences between each instance and its group's mean, of the distances between it and all of its member instances. The clustering process then begins anew, continuing until the centroids do not change their coordinates between successive iterations. This optimal state is referred to as convergence, and its attainment suggests that the centroids are, in fact, in the best possible location, reflected in the minimized SSE.

1.1 Problem Definition

If you consider that an iteration of the distance calculations required to compare each instance to each centroid takes constant time, there are, then, 2 aspects of the k-means algorithm that can be, and often are, targeted for optimization:

1. Centroid initialization, such that the initial cluster centers are placed as close as possible to the optimal cluster centers
2. Selection of the optimal value for k (the number of clusters, and cen-

troids) for a particular dataset

If nothing of the data is known ahead of time, the optimal value of k is typically found via experimentation, though there are algorithms in existence for assisting this process. Centroid initialization is usually performed via randomization, where the space of all existing instances are found, and centroids are randomly placed within this space, with the hopes that the speed and simplicity of this operation is an acceptable trade-off between the provided accuracy and overall performance.

The actual clustering method of the k-means algorithm, which follows centroid initialization, and which is iterated upon until the "best" centroid location is found, is highly sensitive to the initial placement of centroids. The better the initial centroid placement, generally the fewer iterations are required, and the faster and more efficient the algorithm is [4]. Research into improved centroid initialization has yielded both clever methods and improved results, but these methods are often resource intensive and time-consuming.

There are also additional modifications to k-means, such as k-means++ [6], which take a somewhat different approach; the independent steps of centroid initialization and optimal k selection are confounded into a single, overarching process which is iterated upon during the clustering method. This allows the number of clusters, and the corresponding centroids, to fluctuate during the clustering process itself. Again, this process can be resource intensive and time-consuming, and continues to rely on randomization as a main driver of centroid initialization.

What if the trade-off between computational efficiency and the effectiveness of centroid initialization methods was moot? What if there was a deterministic approach which was computationally simple enough that it relied on very few calculations, dramatically reducing the initialization time, lowering the number of clustering iterations required, and shortening the overall execution time of the k-means algorithm, all while producing comparable accuracy of results?

This research proposes a family of such centroid initialization algorithms for k-means clustering which aim to fill this role by using an innovative method for attempting to place the initial cluster centers as close as possible to the optimal cluster centers, and to place them in the same, calculable position every time.

1.2 Proposed Method

The dataset sharding family centroid of initialization algorithms is simple, relying on but a few arithmetic calculations of simple statistical metrics to determine. The algorithm works only for fully numeric datasets; however, as performing distance-based clustering on datasets with nominal values produces meaningless results - true k-means-style clustering of categorical or nominal data requires a related, yet distinct, algorithm such as k-**modes** clustering [7] - this is actually not a limitation.

The sharding family of centroid initialization algorithms proceed in a very simple manner, and are primarily dependant on the calculation of a composite value reflecting all of the attribute values of an instance. The algorithm family supports the calculation of this composite value based on the sum of all instance attribute values (*naive sharding*), the mean of all instance attribute values (*mean sharding*), and the median of all instance attribute values (*median sharding*).

Once this composite value is computed, it is used to sort the instances of the dataset. The dataset is then horizontally split into k pieces, or *shards*. Finally, the original attributes of each shard are independently summed, their mean is computed, and the resultant collection of rows of shard attribute mean values becomes the set of centroids to be used for initialization.

1.3 Expected Results

Putting aside any potential effect on the *overall* k-means algorithm for a moment, such as a reduction in the number of clustering iterations required for convergence, the sharding family of centroid initialization algorithms is expected to exhibit the following beneficial characteristic: decreased centroid initialization time. Sharding initialization is expected to execute much quicker than random centroid initialization, especially considering that the time needed for randomly initializing centroids for increasingly complex datasets (more instances and more attributes) grows superlinearly, and can be extremely time-consuming given complex enough data; sharding initialization executes in linear time, and is not dependent on data complexity.

It is expected that sharding centroid initialization methods will reduce the overall k-means clustering algorithm runtime by reducing the centroid initialization time, as outlined above. Further, it is expected that the sharding family of centroid initialization algorithms will also allow for a further reduction in the overall k-means clustering algorithm runtime by decreasing both the required number of clustering iterations, due to initial centroid placement closer to the optimal, and the overall execution time required by the clustering iterations prior to convergence.

1.4 Thesis Outline

The remainder of this paper will further outline the sharding family of centroid initialization algorithms, define a practical implementation of the algorithms, outline a stringent method for evaluating their performance, and analyze the results of experiments run on the implemented algorithms.

Chapter 2 will outline previous attempts at improving the k-means clustering centroid initialization algorithm. Chapter 3 will describe, in detail, the design of the sharding family of centroid initialization algorithms. Chapter 4 will bridge the gap between the theoretical algorithm design and its practical implementation in Python, including steps that are language-specific. Chapter 5 will share and discuss the results of experiments on the Python implementation of the sharding family of centroid initialization algorithms. Finally, Chapter

6 will state conclusions and discuss potential future work.

Chapter 2

Previous Work

Before describing and implementing an innovative method for centroid initialization for k-means clustering, it would be informative to outline some research which has previously been pursued in an attempt to improve the original initialization process. In this chapter we will investigate a sampling of both improvements and alternatives to random centroid initialization.

While some approaches which challenge the limits of classical k-means clustering will be presented, algorithms which are definitely not k-means will be avoided, of which there are many different types; hierarchical, density-based, and grid-based methods are all families of clustering algorithms which are quite different than k-means clustering. Though k-means is a partitioning clustering algorithm [3], it is certainly not the only one, with other options in this category existent as well. Clustering algorithms such as Gaussian Expectation-Maximization, Fuzzy k-means, and k-harmonic, along with many others, are eliminated from investigation for these reasons.

2.1 Improvements to Random Initialization

A number of attempts at improving randomized centroid initialization have been pursued over the years. A sampling of such research follows.

One such modification to random centroid initialization is k-means++. Arthur & Vassilvitskii [6] recognized that maximizing the distribution of the initial clusters, as opposed to initializing them all at random, was likely advantageous. As such, the approach that their research took was to initialize centroids at random from the data points while weighing potential centroids by their squared distance from the closest centroid already initialized. This effectively ensures a maximal "spread" of centroids across the data space, more so than does randomized initialization.

Arthur & Vassilvitskii found that their approach was, in fact, advantageous. Experiments on 4 datasets yielded results which generally outperformed classical k-means centroid initialization in both accuracy and speed, and often by a wide margin.

Another such exemplar is bisecting k-means, somewhat of a hybrid method which attempts to successively initialize the required number of centroids between iterations of clustering. Bisecting k-means starts with a single centroid and cluster, and then splits this cluster into 2, performs clustering with a value of $k = 2$, and continues to iterate on splitting the best possible target cluster and performing another round of clustering with $k + 1$ until the desired number of clusters (and centroids) are reached.

Steinbach, Karypis, & Kumar [8] found that bisecting k-means generally outperformed standard k-means clustering, when measuring entropy, and that even when it did not it performed nearly as well. Their experiments did not include time-related metrics.

2.2 Alternatives to Random Initialization

Alternatives to randomized centroid initialization, more consistent with the method proposed in this research, have also previously been attempted. Several promising deterministic methods for centroid initialization have emerged from research over the years.

One such deterministic method is found in Su & Dy [9], which outlines a divisive hierarchical method, based on Principal Component Analysis, for centroid initialization. Principal Component Analysis (PCA) is a statistical technique which transforms data into a representative projection in lower-dimensional space. The *principal components* are those variables which descendingly account for the highest amount of variability within the data. PCA also "minimizes the average projection cost, defined as the means squared distance between the data points and their projections" [4], which intuitively sounds as though these principal components, in descending order of their discovery, would make ideal candidates for clusters within the dataset.

Su & Dy leverage this very notion; the results of their experiments on 5 datasets suggest that their implementation led to the k-means clustering process generating clusters with SSE values close to the minimum SSE values obtained by 100 random initialization tests. Results also indicated that fewer numbers of iterations were required for convergence, when compared with random initialization.

Gingles & Celebi [10] undertook research which posited that the natural clusters of a dataset would be located in the areas of highest data density. Since a histogram is deemed the simplest way to estimate nonparametric density, Gingles & Celebi partitioned all attributes of a given dataset into bins and observed the number of values which fell into each of these bins. It could then easily be determined which areas of the dataset were of the highest density.

Results of [10] showed that this implementation generally performed well in terms of mean sum of squared error metrics, when compared with other deterministic k-means centroid initialization methods on normalized datasets. The study did not, unfortunately, draw comparisons between the outlined innovative method and non-deterministic initialization methods, such as randomized; nor did the study use any time-related metrics. The results do, however, express that a simple, arithmetic-based deterministic centroid initialization method, not rooted in randomization, could provide favorable outcomes.

No comparable research employing such a simple arithmetic-based and arbitrary composite instance value-based method as this research outlines could be found in the research. However, as non-deterministic centroid initialization algorithms have previously presented promising results, and as related approaches to solving the problem of reduced k-means clustering centroid initialization times with comparable clustering algorithm sum of squared error results are not identifiable, the suggestion is that this research is worthy of pursuit.

Strathmore
PURE COTTON

Chapter 3

Methodology

This chapter will further outline the functionality of the proposed method of centroid initialization, and serve to highlight the differences between it and random initialization, in the context of k-means clustering.

The first task is to formally define the k-means clustering algorithm and its constituent components. Once these algorithms are sufficiently recounted, the proposed centroid initialization method will be outlined in detail.

3.1 The k-means Clustering Algorithm

The concept of clustering has been previously presented with the goal of "maximizing the intraclass similarity and minimizing the interclass similarity" [3]. While there are numerous clustering algorithms in existence, varying in their complexities, k-means takes a rather direct approach in its attempts to achieve this stated goal.

k-means is a simple, yet often effective, approach to clustering. k points are randomly chosen as cluster centers, or *centroids*, and all training instances are plotted and added as a member of the closest centroid's cluster. After all instances have been added to clusters, the centroids, representing the means (the **means** in k-means) of the collection of each cluster's instances, are re-calculated, with these re-calculated centroids becoming the new centers of their respective clusters.

At this point, all cluster membership is reset, and all instances of the training set are re-plotted and -added as members of the closest, possibly re-centered (or different) cluster. This iterative process continues until there is no change to the centroids or their membership, at which point the clusters are considered settled [3], and convergence has been achieved.

The classical k-means clustering algorithm is presented as *Algorithm 1*.

Convergence is achieved once the re-calculated centroids match the previous iteration's centroids. The measure of distance utilized by the k-means clustering algorithm is generally Euclidean, which, given 2 points in the form of *(x, y)*, can be represented as:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}. \tag{3.1}$$

This equation can be generalized to any number of points; therefore, the Euclidean distance between any data instance and a centroid is calculable by

Algorithm 1 k-means Clustering

```
INPUT: $S$, $k$ where $S$ = set of instances, $k$ = integer
OUTPUT: $k$ Clusters
Require: $S \neq \varnothing$, $k > 0$
procedure ClusterData:
    $C \leftarrow$ InitializeCentroids($S$, $k$)
    repeat
        for all Instance $i$ in $S$ do
            $shortest \leftarrow 0$
            $membership \leftarrow null$
            for all Centroid $c$ in $C$ do
                $dist \leftarrow$ Distance($i$, $c$)
                if $dist < shortest$ then
                    $shortest \leftarrow dist$
                    $membership \leftarrow c$
                end if
            end for
        end for
        UpdateCentroids($C$)
    until convergence
end procedure
```

taking the square root of the sum of the element-wise squared difference of attribute value vectors representing a particular dataset instance and a given centroid. This generalization can be represented as:

$$\sqrt{(a_{1_i} - a_{1_c})^2 + (a_{2_i} - a_{2_c})^2 ... (a_{n_i} - a_{n_c})^2}, \tag{3.2}$$

where a is a given attribute, i and c denote instance and centroid attribute values, respectively, and 1, 2, and n represent the particular attribute index for which the difference is being calculated.

3.2 Random Centroid Initialization

Classical k-means clustering utilizes random centroid initialization [3]. In order for randomization to be properly performed, the entire space being occupied by all instances in all dimensions is first to be determined. This means that all instances in the dataset must be enumerated, and a record must be kept of the minimum and maximum values of each attribute, constituting the boundaries of the various planes in n-dimensional space, with n being the number of attributes in the dataset. This can be a time-consuming process; the time required to perform randomization grows exponentially with increased dataset complexity, by way of both a larger number of instances and a larger number of attributes.

The algorithm for random centroid initialization is presented as *Algorithm*

2. Note that the **RandomCentroids** function of *Algorithm 2* is generally the **InitializeCentroids** function being referenced in *Algorithm 1*, at least in classical k-means; however, random centroid initialization is not a mandatory component of the k-means clustering process.

Algorithm 2 Random Centroid Initialization

```
INPUT: $S$, $k$ where $S = set\ of\ instances$, $k = integer$
OUTPUT: $k\ Centroids$
Require: $S \neq \varnothing$, $k > 0$
procedure RandomCentroids:
    $C \leftarrow$ size($k$)
    for all $Attribute\ a$ in $S$ do
        $min \leftarrow$ Min($a$)
        $max \leftarrow$ Max($a$)
        $range \leftarrow max - min$
        for all $Centroid\ c$ in $C$ do
            $Attribute\ a \leftarrow min + range \times$ Rand$(0, 1)$
        end for
    end for
end procedure
```

After the centroids are initialized, the clustering portion of the algorithm is executed (see *Algorithm 1*). Once an iteration of clustering has taken place, centroids must be re-centered, as described in the above treatment of the k-means clustering algorithm.

Centroid updating is described in *Algorithm 3.*

As outlined previously, this combined iterative process of clustering and centroid updating continues until there is no change in centroid coordinates between successive iterations, at which point the clusters are considered settled, and convergence has been achieved.

Algorithm 3 Centroid Updating

```
INPUT: $C$, where $C$ = set of clusters
OUTPUT: Set of centroids
Require: $C \neq \varnothing$
procedure UpdateCentroids:
    $NewCentroids \leftarrow$ size($C$)
    for all Cluster $c$ in $C$ do
        for all Attribute $a$ in $c$ do
            $mean \leftarrow$ Mean($a$)
            $a$ of Centroid $c \leftarrow mean$
        end for
        Append $c$ to NewCentroids
    end for
end procedure
```

A few aspects of particular importance regarding the k-means clustering algorithm, and its component methods, are presented below.

- Time required for random centroid initialization is affected by the time required to survey the entire dataset in order to determine the dataset-occupied space and its boundaries
- The number of clustering iterations required to reach convergence can vary, dependant on the initial random centroids, which has a direct effect on the total execution time of the clustering algorithm

These particular aspects of the k-means clustering algorithm are those which are directly addressed by the sharding family of centroid initialization algorithms.

Now that the k-means clustering algorithm has been outlined in sufficient detail, the sharding method of centroid initialization will be defined.

3.3 Sharding Centroid Initialization

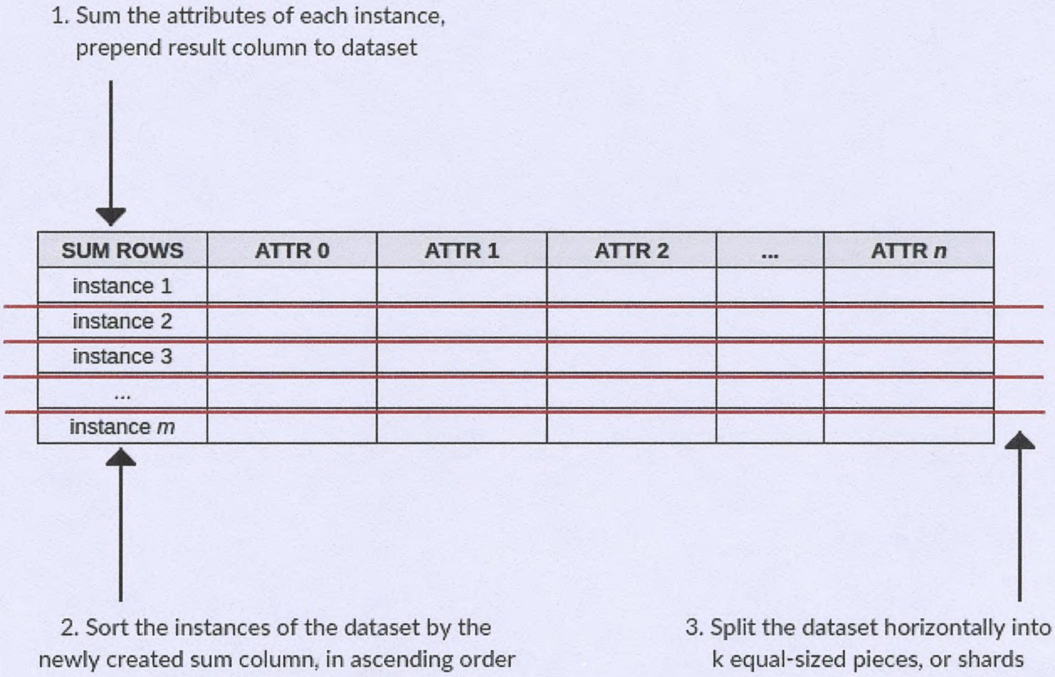
While random centroid initialization takes a non-deterministic approach to initializing the first batch of centroids for k-means clustering, the proposed algorithm family addresses the task in a different manner. The dataset sharding family of centroid initialization algorithms are deterministic in nature, and rely on simple arithmetic for determining the initial set of clusters.

All of the member algorithms operate within a common framework. First, each of the member instances of the dataset to be clustered has some composite value computed from its attributes' values intended to provide some reflection of the instance itself, able to be used for comparison to other instances for sorting. This is the precise step where the constituent methods of the algorithm family differ in their approach, approaches which are detailed below.

Next, as alluded to above, the dataset instances are sorted by this composite value. The sorted dataset is then split horizontally into k equal-sized sections, or *shards*. Finally, the mean of each of the attribute columns of each of k shards is used as the corresponding attribute of the centroid of that shard's cluster.

3.3.1 Naive Sharding

The sharding family of centroid initialization algorithms includes 3 separate statistical methods for performing the first of these steps. *Naive sharding* employs the summation of each of the dataset instance attribute values to achieve



1. Sum the attributes of each instance, prepend result column to dataset

SUM ROWS	ATTR 0	ATTR 1	ATTR 2	...	ATTR *n*
instance 1					
instance 2					
instance 3					
...					
instance *m*					

2. Sort the instances of the dataset by the newly created sum column, in ascending order

3. Split the dataset horizontally into k equal-sized pieces, or shards

Figure 3.1: Sorting by composite value and sharding.

the composite value used for sorting an entire dataset prior to *sharding*.

An overview of naive sharding is as follows:

- **Step 1**. Sum the attribute (column) values for each instance (row) of a dataset, and prepend this new column of instance value sums to the dataset
- **Step 2**. Sort the instances of the dataset by the newly created sum column, in ascending order
- **Step 3**. Split the dataset horizontally into k equal-sized pieces, or shards
- **Step 4**. For each shard, sum the attribute columns (excluding the column created in step 1), compute its mean, and place the values into a

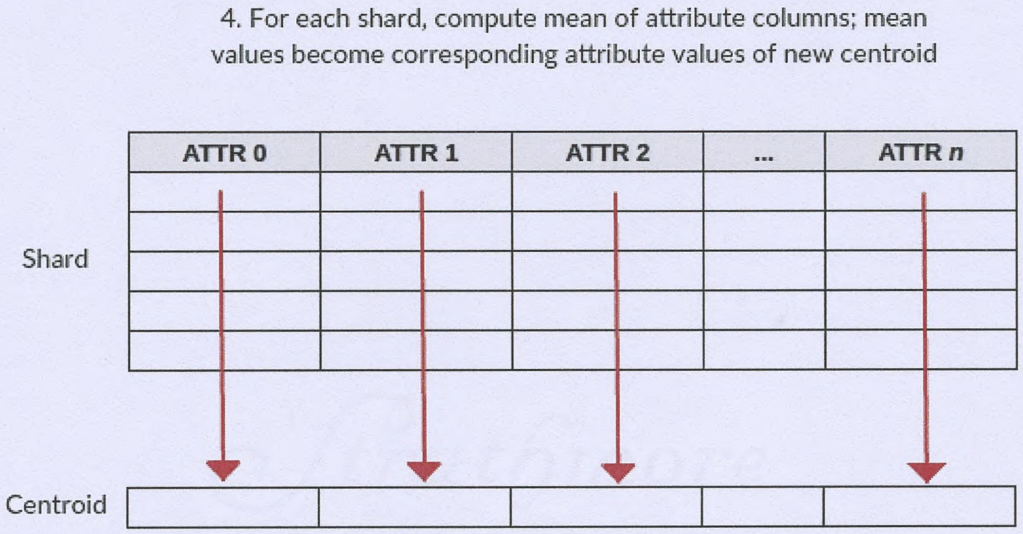

4. For each shard, compute mean of attribute columns; mean values become corresponding attribute values of new centroid
ATTR 0
ATTR 1
ATTR 2
...
ATTR *n*
Shard
Centroid

Figure 3.2: Shard attribute means become centroid attribute values.

new row; this new row is effectively one of the centroids used for initialization

- **Step 5**. Add each centroid row from each shard to a new set of centroid instances
- **Step 6**. Return this set of centroid instances to the calling function for use in the k-means clustering algorithm

As previously promised, the algorithm is very simple.

Formally, the naive sharding centroid initialization algorithm for k-means clustering is described in *Algorithm 4*.

Algorithm 4 Naive Sharding Centroid Initialization

```
INPUT: $S$, $k$ where $S$ = set of instances, $k$ = integer
OUTPUT: $k$ Centroids
Require: $S \neq \varnothing$, $k > 0$
 1: procedure NaiveSharding:
 2:     for all Instance $i$ in $S$ do
 3:         $composite \leftarrow$ Sum($attributes$)
 4:         Append $composite$ to $i$
 5:     end for
 6:     Sort $S$ on $composite$
 7:     Split $S$ into $k$ Shards
 8:     for all Shard $k$ in Shards do
 9:         for all Attribute $a$ in $k$ do
10:             $mean \leftarrow$ Mean($a$)
11:             $a$ of Centroid $k \leftarrow mean$
12:         end for
13:     end for
14: end procedure
```

3.3.2 Mean Sharding

Mean sharding, the second of the sharding family of centroid initialization methods to be outlined, is the same as naive sharding in every aspect, except for the statistical method used to sort the dataset instances prior to sharding. Whereas naive sharding uses a summation process, mean sharding finds the mean of all of the attributes' values and uses this number as the composite value on which to sort.

As such, the algorithm for mean sharding is identical to what is presented in *Algorithm 4*, with the following 4 lines replacing lines 2-5 of *Algorithm 4*. In fact, only a single one of these lines differs from its corresponding line in *Algorithm 4*; the other lines are included to provide context.

Algorithm 5 Mean Sharding Centroid Initialization

```
for all Instance i in S do
    composite ← Mean(attributes)
    Append composite to i
end for
```

3.3.3 Median Sharding

Finally, *median sharding* calculates its composite value for dataset sorting by taking the attribute values of each instance and finding their median. Just as *Algorithm 5* shows only the lines differing from those in *Algorithm 4*, the following does the same. Again, note that only a single line is actually different; the others are provided for context.

Algorithm 6 Median Sharding Centroid Initialization

```
for all Instance i in S do
    composite ← Median(attributes)
    Append composite to i
end for
```

A few aspects of particular importance regarding the sharding family of centroid initialization algorithms, especially in contrast to those pointed out above for the random centroid initialization algorithm of classical k-means, are listed below.

- Time required for sharding centroid initialization is **not** affected by the time required to survey the entire dataset and compute occupied space and boundaries, as only simple arithmetic is performed on instance data in linear time
- The number of clustering iterations required to reach convergence will re-

main constant between executions of the algorithm on the same dataset, as the results of simple statistical calculations will always provide the same results, a fact which stabilizes the total execution time of the clustering algorithm

The practical implementation described in the following chapter, and the consequent experiments on its performance, will provide insight into whether these above aspects of the proposed algorithm lead to improved execution times of the k-means clustering algorithm.

3.4 Measuring Success

This project's measure of success will be the execution time savings of the k-means clustering algorithm when utilizing the sharding family of centroid initialization methods over random centroid initialization. Independent of such performance benefits, it is imperative that any execution time savings are accompanied by reasonably similar measures of cluster algorithm accuracy, which will be evidenced by the comparison of the SSE metric of both random and sharding centroid initialization method outcomes.

If huge increases in time savings are attained, but at a cost of algorithm accuracy to any great degree, the trade-off may not be deemed worthy of consideration. Conversely, increased execution time savings balanced with similar SSE metrics - which is the desired outcome - would almost certainly make the sharding family of centroid initialization algorithms valuable additions to the clustering landscape. The next chapter begins to explore this possibility.

Chapter 4

Implementation

This chapter provides the details of translating the methodology outlined in the previous chapter into code for evaluating the proposed algorithms. As Python is a solid default language for contemporary scientific computing and machine learning implementations [11], and due to its deserved reputation as being easy to both read and write, it has been chosen as the medium for the practical portion of this research.

First, a brief discussion of the scientific Python computing ecosystem and its common libraries is undertaken. The chapter is then broken down into sections corresponding to those in the previous chapter, allowing for a straightforward comparison of the theoretical proposed algorithms and their practical implementation. Python code is presented for aspects of the implementation which are deemed critical, which represent the proposed functionality, or which require further clarification.

4.1 Python

Python[1] is an open source, general-purpose high-level programming language which supports multiple paradigms and enjoys an admirable market share [12]. Beyond this, Python is heavily used in the scientific computing community, has made recent inroads into statistical computing, and is increasingly becoming the language of choice for machine learning researchers and practitioners everywhere [11]. Given that it is open source, and has been described as *executable pseudocode* [13], adoption by those looking for a cooperative and flexible language with quick prototyping turnaround times is intuitive. In addition to these characteristics supporting do-it-yourself algorithm implementations, from classical machine learning algorithms to contemporary deep neural networks and beyond, Python also has community-developed and -maintained (and often considered *de facto*) libraries for nearly any machine learning or data mining task imaginable.

According to longstanding data mining and analytics information website KDnuggets' annual survey of analytics and data mining tools[2], Python has made continual market share inroads over the past several years in the analytics, data science, and machine learning communities. Details of the survey methodology can be found on KDnuggets' website.

Figure 4.1 shows the previous 6 years of Python market share data, according to KDnuggets. There is a clear upward trend, not unlikely due to the numerous

---

[1] https://www.python.org/

[2] http://www.kdnuggets.com/2015/05/poll-r-rapidminer-python-big-data-spark.html

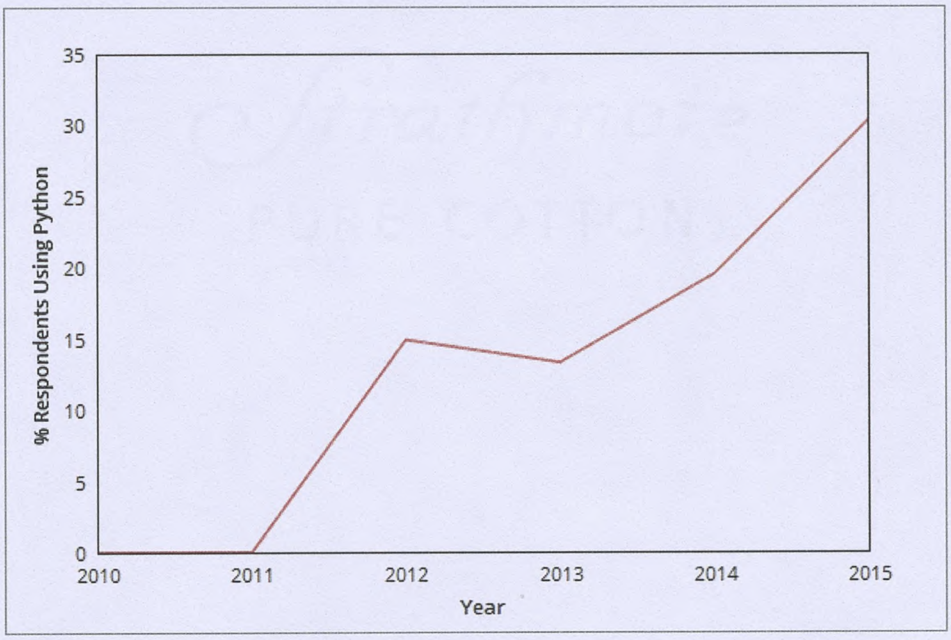


Figure 4.1: Python popularity among the analytics community.

language characteristics previously highlighted. Note that the survey is not scientific, and is simply intended to provide some insight into Python's role in the current analytics and machine learning landscape.

Numerous well-known and -utilized scientific computing libraries exist for Python as well. Notably, the **NumPy**[3] scientific library, and its powerful N-dimensional array (matrix) objects, is relied upon for this research as a dataset housing and manipulation mechanism. NumPy is highly-optimized Python and C code wrapped in a Python application programming interface (API) [14], providing both the lower-level resource management and the speed and convenience that come with such an implementation architecture.

Beyond NumPy, particular modules of the closely-related **SciPy**[4] superset library for scientific computation have been employed, specifically for vectorized distance calculations during the clustering process [14]. NumPy and SciPy provide well-honed and high-performing implementations of the functionality required in each of the disclosed tasks; re-inventing the wheel would seem not only clunky and unnecessary for this research, not removing every possible point of interference with the true algorithmic aspects which are being explored by this research would seem both dishonest and counterproductive.

---

[3] http://www.numpy.org/

[4] https://www.scipy.org/

Additionally, 2-dimensional Python plotting library **matplotlib**[5] has been employed for the plots and graphs which have been produced in the results analysis section of this paper.

4.2 Implementation Overview

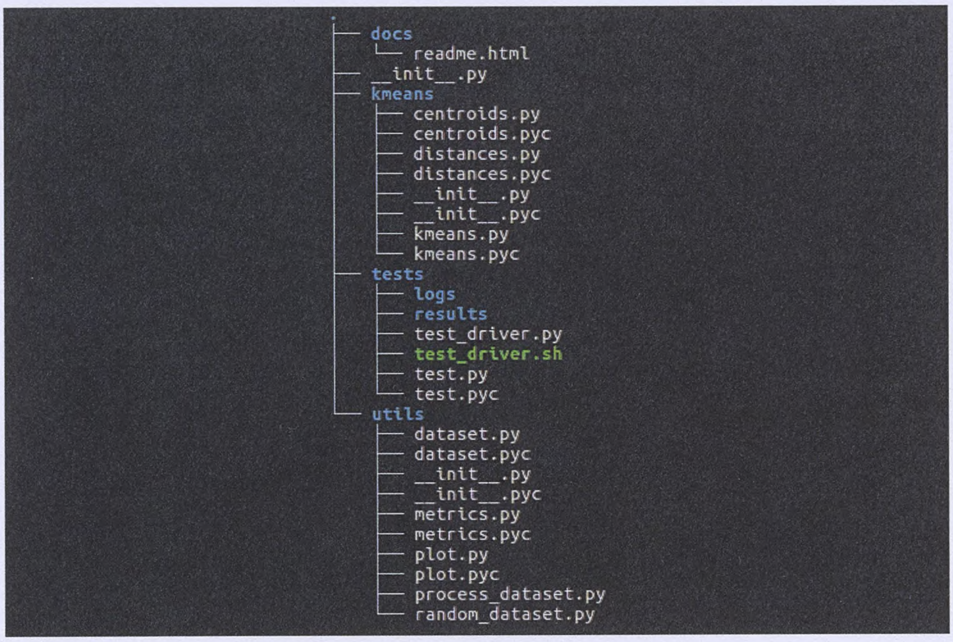
For this research, the k-means clustering algorithm, including the random centroid initialization method and the sharding family of centroid initialization methods, as well as utility functionality and much of the testing apparatus, has all been implemented in Python. *Figure 4.2* is an outline of the project structure, including relevant packages and modules.

The following is a project package overview:

- **docs** - includes project documentation
- **kmeans** - the k-means algorithm and associated algorithmic functionality

---

[5] http://matplotlib.org/



```
.
├── docs
│   └── readme.html
├── __init__.py
├── kmeans
│   ├── centroids.py
│   ├── centroids.pyc
│   ├── distances.py
│   ├── distances.pyc
│   ├── __init__.py
│   ├── __init__.pyc
│   ├── kmeans.py
│   └── kmeans.pyc
├── tests
│   ├── logs
│   ├── results
│   ├── test_driver.py
│   ├── test_driver.sh
│   ├── test.py
│   └── test.pyc
└── utils
    ├── dataset.py
    ├── dataset.pyc
    ├── __init__.py
    ├── __init__.pyc
    ├── metrics.py
    ├── metrics.pyc
    ├── plot.py
    ├── plot.pyc
    ├── process_dataset.py
    └── random_dataset.py
```

Figure 4.2: Project tree structure.

- **tests** - apparatus for performing the implementation experiments
- **utils** - utility modules, including those for dataset management, plotting, and algorithm performance metrics

For information on executing the implemented algorithm, see the project's documentation, browse the inline comments throughout the code, or read the testing modules code.

Note that the script used for driving the experiments is the only project code not written in Python; it is implemented in shell script for purposes described in a subsequent section. Consider that this code was implemented in a Linux environment, and as such may not behave as expected elsewhere.

4.3 *k*-means Clustering Algorithm

The actual *k*-means clustering algorithm, including all of its constituent components, such as distance calculation and centroid initialization functions, is implemented in the *kmeans* package; in particular, much of the interesting functionality of this research is located in the *centroids.py* module.

Listing 4.1 is an excerpt from the *kmeans.py* module.

Listing 4.1: *kmeans.py* module excerpt.

```
# Loop until no changes to cluster assignments
while changed:

  # Change in clusters between iterations?
  changed = False

    # For every instance (row in dataset)
    for i in range(m):

      # Track minimum distance, and vector index of cluster
      min_dist = np.inf
      min_index = -1

      # Calculate distances
      for j in range(k):
        dist_ji = dist_function(cents[j,:], ds[i,:])
        if dist_ji < min_dist:
          min_dist = dist_ji
          min_index = j

      # Check if cluster assignment of instance has changed
      if cluster_assignments[i,0] != min_index:
        changed = True

      # Assign instance to appropriate cluster
      cluster_assignments[i,:] = min_index, min_dist**2
```

```
# Update centroid locations
for cent in range(k):
  points = ds[np.nonzero(cluster_assignments[:,0] \
    .A==cent)[0]]
  cents[cent,:] = np.mean(points, axis=0)
```

This piece of code performs the actual clustering task of k-means clustering, which occurs after centroids have been initialized, randomly or otherwise, and executes until convergence has occurred. For each instance in the dataset, the distance to each centroid is calculated, and the closest centroid, and, logically, member cluster, is tracked. Afterward, whether or not there has been a change in assigned cluster for the given instance is determined (the convergence criteria), and the cluster assignment, and its squared distance, is recorded. Finally, the centroid locations are updated before beginning the next iteration, should the convergence criteria not have been met.

The Euclidean distance of 2 points, generalized, for this research, to any number of corresponding attribute value vectors between a centroid and a dataset instance, has been implemented using the distance functions of SciPy. As such, the code is straightforward, being called from line 16 of the above *cluster()* function as *dist_function()*. This is actually an abstraction, a variable referencing any valid function call, in the case that something unorthodox, such as Manhattan (city block) or Minkowski distance, was desired to be tested with this code. Such an avenue has not been pursued for this research, however.

The actual code excerpt which performs the SciPy distance function call is shown is *Listing 2*.

Listing 4.2: *euclidean()* function from *distances.py* module.

```
import numpy as np
import scipy.spatial.distance as metric

def euclidean(A, B):
  # Call to scipy with vector parameters
  return metric.euclidean(A, B)
```

4.4 Random Centroid Initialization

Random centroid initialization is accomplished via the *rand_cent()* function of the *centroids.py* project module. The function accepts 2 input parameters, a dataset, *ds*, as a NumPy matrix and an integer, *k*, representing the number of centroids to create, and proceeds by first surveying the bounded data space and then randomly selecting points within this space. It is evident that this process actually happens attribute value by attribute value. The set of centroids is returned as a NumPy matrix.

Listing 4.3: *rand_cent()* function from *centroids.py* module.

```
def rand_cent(ds, k):

  # Number of columns in dataset
  n = np.shape(ds)[1]

  # The centroids
  centroids = np.mat(np.zeros((k,n)))

  # Create random centroids
  for j in range(n):
        min_j = min(ds[:,j])
        range_j = float(max(ds[:,j]) - min_j)
        centroids[:,j] = min_j + range_j * np.random.rand(k, 1)

    # Return centroids as numpy array
    return centroids
```

The function shown in *Listing 4.3* is called from the relevant section of code in the *cluster()* function of the *kmeans.py* module via abstraction in a similar manner as the *dist_function()* function, which allows a user to pass the desired centroid initialization function to the *cluster()* function as an input parameter, without the need for code specific to calling each of the centroid initialization functions.

Figure 4.3 is a diagrammatic representation of the interaction between the constituent components of the *kmeans* module.

It is to be noted that the code included thus far has been inspired and influenced by Harrington [15].

4.5 Sharding Centroid Initialization

The following sections outline the implementation of the sharding family of centroid initialization functions.

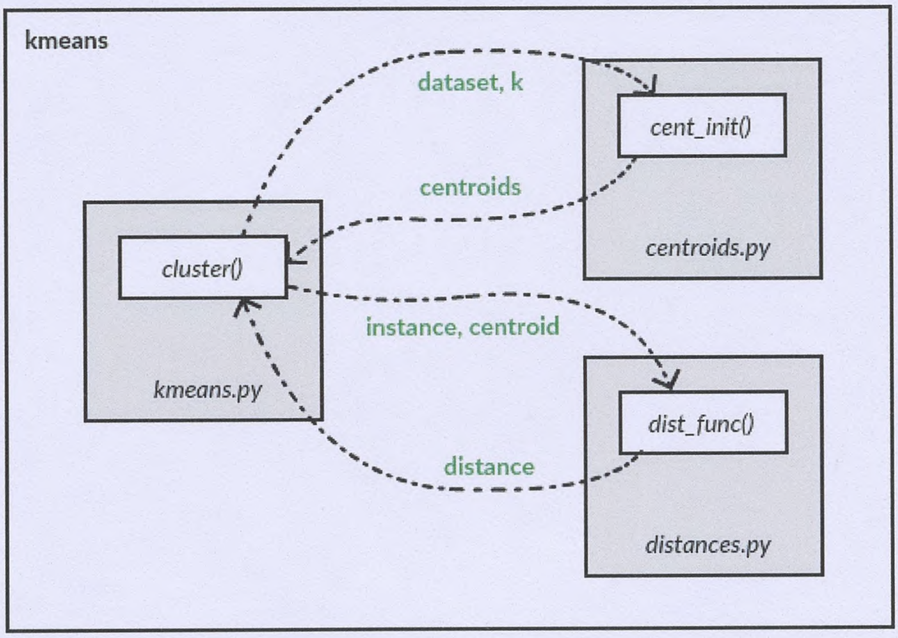


Figure 4.3: *kmeans.py* module overview.

4.5.1 Naive Sharding

As outlined in a previous section, the naive sharding centroid initialization method proceeds using the following steps:

- **Step 1**. Sum the attribute (column) values for each instance (row) of a dataset, and prepend this new column of instance value sums to the dataset
- **Step 2**. Sort the instances of the dataset by the newly created sum column, in ascending order
- **Step 3**. Split the dataset horizontally into k equal-sized pieces, or shards
- **Step 4**. For each shard, sum the attribute columns (excluding the col-

umn created in step 1), compute its mean, and place the values into a new row; this new row is effectively one of the centroids used for initialization

- **Step 5**. Add each centroid row from each shard to a new set of centroid instances
- **Step 6**. Return this set of centroid instances to the calling function for use in the k-means clustering algorithm

Listing 4.4: *naive_sharding()* function from *centroids.py* module.

```
def naive_sharding(ds, k):

  # Number of columns in dataset
  n = np.shape(ds)[1]

  # Number of rows in dataset
  m = np.shape(ds)[0]

  # The centroids
  centroids = np.mat(np.zeros((k,n)))

  # Sum all elements of each row
  composite = np.sum(ds, axis=1)

  # Add composite as first column to original dataset matrix
  ds = np.append(composite, ds, axis=1)

  # Sort matrix based on sum_rows (first) column
  ds.sort(axis=0)

  # Step value for dataset sharding
  step = floor(m/k)
  # Vectorize mean ufunc for numpy array
  vfunc = np.vectorize(_get_mean)

  # Shard dataset; compute centroids
  for j in range(k):
    if j == k-1:
      centroids[j:] = vfunc(np.sum(ds[j*step:,1:], \
        axis=0), step)
    else:
      centroids[j:] = vfunc(np.sum(ds[j*step:(j+1)*step,1:], \
        axis=0), step)

  # Return centroids as numpy array
  return centroids
```

For more information on the theoretical methodology of the naive sharding centroid initialization method, see *Algorithm 4*, *Figure 3.1*, and *Figure 3.2* of chapter 3.

Listing 4.4 implements the prescribed methodology in practical form. The comments outline the step-by-step process of the algorithm, which match up almost verbatim with the steps above. Of note, line 13 calculates the representative composite value for each instance, used for subsequent sorting (line 19) before dataset sharding and centroid computation (lines 28-34). Line 25 vectorizes a custom helper function (*_get_mean()*), making it useful for NumPy arrays as opposed to only traditional scalars, for finding the shard means, which is outlined in the code of *Listing 4.5*.

Listing 4.5: *_get_mean()* function from *centroids.py* module.

```
def _get_mean(sums, step):
  # Return means as numpy array
  return sums/step
```

4.5.2 Mean Sharding

For mean sharding, a single line must be replaced in the above naive sharding algorithm in order to make the necessary changes, computing a composite value reflective of the instance means as opposed to their sums. With NumPy, this is trivial. Line 13 of *Listing 4.4* is replaced by the single line of code of *Listing 4.6*.

Listing 4.6: *mean_sharding()* function excerpt from *centroids.py* module.

```
# Get mean of all elements of each row
composite = np.mean(ds, axis=1)
```

The remainder of the algorithm follows *Listing 4.4*.

4.5.3 Median Sharding

For median sharding, the final member of the sharding family of centroid initialization methods, the same single line, 13, from *Listing 4.4* can be replaced with the following single line of code of *Listing 4.7*. Again, this is trivial using NumPy.

Listing 4.7: *median_sharding()* function excerpt from *centroids.py* module.

```
# Get median of all elements of each row
composite = np.median(ds, axis=1)
```

4.6 Utility Modules

Aside from core project functionality, code for this research includes a small number of utility modules, which are outlined below.

- ***dataset.py*** - functionality for loading a dataset from comma-separated value (CSV) file
- ***metrics.py*** - functionality for computing the sum of squared errors (SSE) for a particular cluster
- ***plot.py*** - functionality for plotting a 2-dimensional representation of a dataset and a set of centroids
- ***process_dataset.py*** - used for processing datasets in a variety of ways, such as removing instances with empty values, or removing entire columns; used for dataset preparation outlined in a subsequent section
- ***random_dataset.py*** - used to generate an artificial dataset of random values; employed in the creation of datasets outlined in a subsequent section

4.7 Testing Apparatus

The following provides an overview of the testing modules and process followed for conducting experiments in this research.

- **/results** - output from each experiment, including a series of parameters such as the number of clusters for the experiment, number of k-means clustering iterations, mean SSE of all clusters for the experiment, and a number of timing-related measures

- **/logs** - raw output of the report data the k-means clustering implementation outputs in verbose mode, including mostly the same data as above, except per iteration of the k-means clustering algorithm, as opposed to summarized per experiment
- ***test.py*** - test script which executes a single k-means clustering task
- ***test_driver.sh*** - shell script which automates the execution of a number of k-means experiments and captures their results, configured to execute all experiments required of a single dataset, including those for random, naive, mean, and median sharding

The following chapter describes the results of the experiment executions.

Chapter 5

Results and Evaluation

This chapter describes and evaluates the results of the experiments executed against the implementation described in the previous chapter.

First, an overview of the experiments performed are presented. The following section then describes the datasets which were used for experimentation, presents the outcomes of those experiments, and discusses relevant aspects of those experiment outcomes. These outcomes are discussed with an eye toward pivoting to conclusions in the final chapter.

5.1 Experiment Overview

For each of the datasets described in the following section, a set of experiments was executed. These experiments focused on executing the k-means clustering algorithm using the previously outlined centroid initialization methods -

random, naive sharding, mean sharding, median sharding - with an interest in capturing and comparing their sum of squared error (SSE) metric outcomes, their execution times (both for the entire clustering execution as well as for the centroid initialization itself), and the number of iterations of clustering required for convergence.

While the execution times and number of iterations for convergence are a direct and obvious measure of the effectiveness of the centroid initialization algorithms, SSE may not be. Should a member of the sharding family of centroid initialization algorithms lead to a clustering performance with a significantly reduced execution time yet demonstrate SSE measure values which were not comparable to random initialization on the same data, it would be difficult to conclude that this was any type of improvement. There may be some possibility of trade-off between speed and accuracy in general; however, terribly inaccurate results are not an acceptable outcome in exchange for dramatic improvements in speed.

For each of the datasets, the following individual experiments were executed:

- Random centroid initialization - 10 consecutive executions
- Naive sharding centroid execution - 3 consecutive executions
- Mean sharding centroid execution - 3 consecutive executions
- Median sharding centroid execution - 3 consecutive executions

As random centroid initialization leads to different initial centroids on each execution, it is reasonable to believe that performance outcomes may differ significantly; in fact, they often do [3]. However, for the deterministic sharding family of centroid initialization algorithms, initial centroids are always the same for a given dataset, leading to identical SSE outcomes, and so fewer runs are required to simply determine mean execution times. Each set of experiments were executed on the same dedicated computer (see *Appendix A* for details).

Python's *clock()* [16] function of its built-in *time* module was employed for capturing the runtimes of the separate clustering and centroid initialization algorithm executions, as these functions are executed from within Python code and return values to the calling functions, making this the most appropriate technique for performing this task. As such, values reported in the results tables below are the difference between wall clock times, in seconds.

For the overall execution time of each k-means clustering experiment, however, the *time* [17] command of the GNU operating environment was utilized, in order to isolate the resources dedicated exclusively to our process(es); as these executions were called from outside of the Python environment, this technique was appropriate for such a task. As such, values reported in the results tables below are in CPU seconds.

Listing 5.1: Timing functionality excerpt from *kmeans.py* module.

```
# Initialize centroids
t0 = time.clock()
cents = cent_init(ds, k)
t1 = time.clock()
cent_init_time = t1 - t0
```

Timing was captured within Python as outlined in *Listing 5.1*, with this particular example tracking the runtime of the centroid initialization function.

The full testing code apparatus has been detailed in the previous chapter.

5.2 Datasets

For some datasets, particularly those for which the number of natural clusters was unknown, the experiments were performed a number of times, each with a different value of k (the number of desired clusters), in order to provide maximum insight.

For datasets for which the number of natural clusters was known, one iteration of the experiment was performed, with an appropriate value of k. For datasets for which the number of natural clusters was known, and for which a dataset class attribute was available, the experiment was generally performed twice; one execution with the class attribute removed, and the other with the class attribute intact, converted to a numeric representation if necessary. While clustering with values for which distances between cannot be meaningfully measured (this generally includes nominal class values) is generally meaningless and adds noise to a dataset, the process was undertaken as a comparative measure between the centroid initialization algorithms, as opposed to an exercise in evaluating the results of the actual effectiveness of k-means clustering; i.e. the interest is in the relative, as opposed to the absolute.

The datasets are of differing sizes, complexity, and data distribution; as such, they are grouped into like categories. Each dataset is described in appro-

Table 5.1: Dataset summary.

Dataset	Instances	Attributes	Clusters
Dessewffy	100	2	n/a
Jelonek	1000	2	n/a
Ruspini	75	2	4
Iris	150	4	3
Wine	178	13	3
Seeds	210	7	3
3D Road Network	434874	4	n/a
Power Consumption	2049280	9	n/a

priate detail, and the results of the above experiments for said dataset are summarized alongside.

For experimentation, 8 datasets have been employed. Two of these sets have been artificially crafted for this research, while the others are well-utilized sets from the research.

Table 5.1 summarizes the datasets used, with attributes reported *not* including class attributes, should one exist for the given dataset.

5.2.1 Research-specific Artificial Datasets

This section includes the artificial datasets crafted specifically for this research, having been created by the researcher with specific characteristics in mind, characteristics which are outlined below. These 2 datasets, Dessewffy and Jelonek, have been named using the *randomized Wikipedia article*[1] *naming system*; for more information, see *Appendix 2.*

---

[1] https://en.wikipedia.org/wiki/Wikipedia:Random

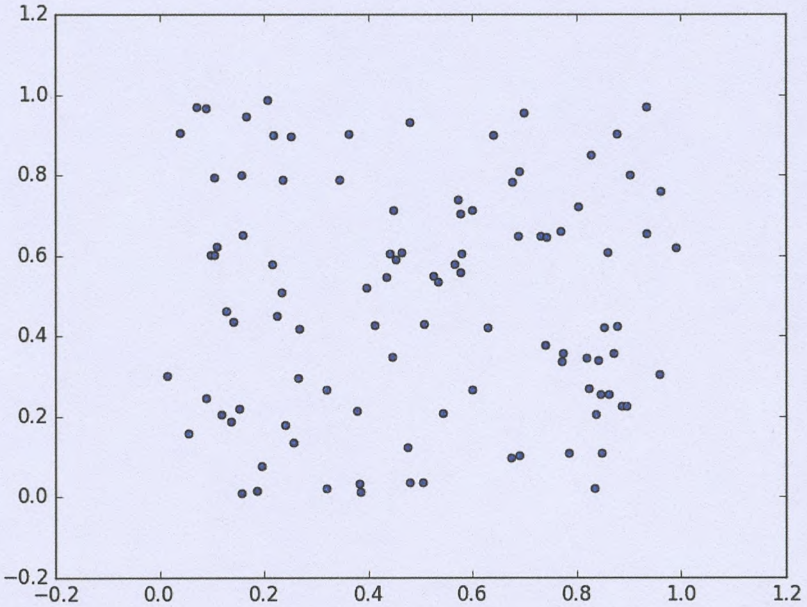


Figure 5.1: Dessewffy dataset.

Dessewffy

The Dessewffy dataset is an artificial dataset created by the researcher. The dataset is uniformly randomized in nature, consisting of 2 attributes and 100 instances. A very simple dataset, Dessewffy mimics 2-dimensional Euclidean space, and, as such, is informative in that its structure can be easily visualized by the human eye.

The randomized uniform distribution will also provide an additional layer of insight into the sharding family of centroid initialization algorithms, given that there will most certainly not be any patterns of note held within this dataset. Given its simple nature, it is a good set of data to begin experimentation

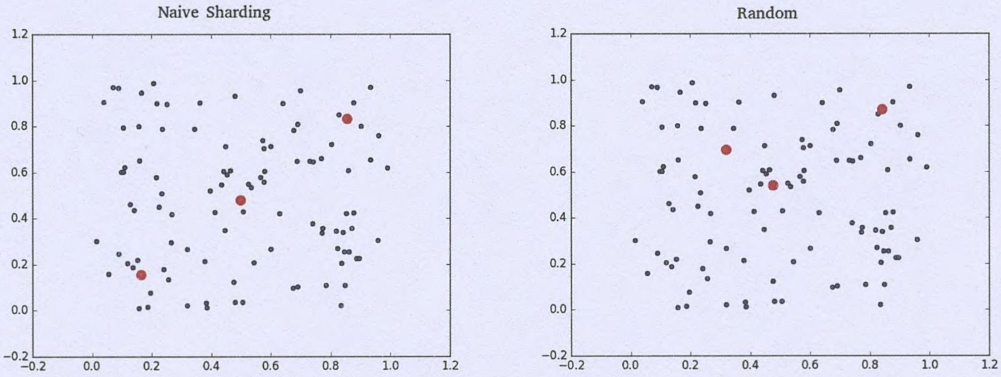


Figure 5.2: Dessewffy naive sharding versus random initialization, $k = 3$.

with.

Note that both the Dessewffy and Jelonek datasets were created with the *random_dataset.py* script located in the *utils* module directory of the project.

The Dessewffy dataset is visualized in *Figure 5.1.*

In order to visualize the difference between random centroid initialization and sharding initialization, *Figure 5.2* demonstrates where one particular execution of random centroid initialization places 3 centroids in relation to the Dessewffy data instances, alongside where naive sharding places its initialization centroids, each and every time.

Figure 5.3 demonstrates random versus naive sharding centroid initialization for $k = 7$.

Experiments for Dessewffy included executing the testing regimen outlined above for k values of 3, 7, and 10. The results of these executions are outlined in *Table 5.2.*

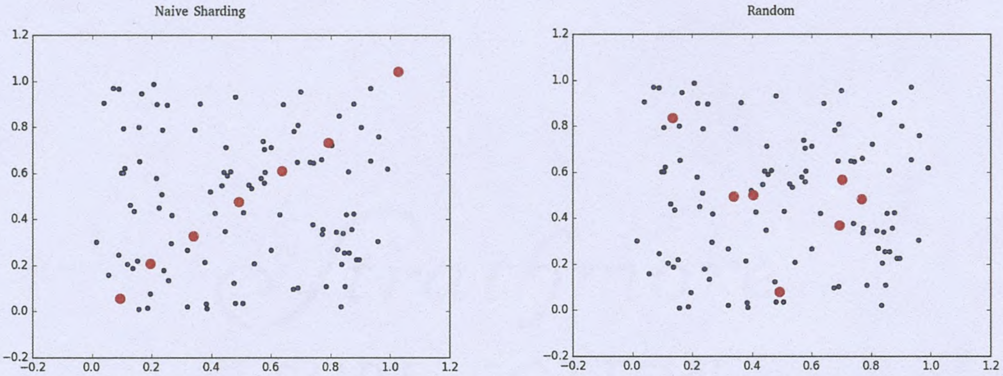


Figure 5.3: Dessewffy naive sharding versus random initialization, $k = 7$.

Table 5.2: Dessewffy results.

k	Init Type	Iters	Mean SSE	Init Time	Clust Time	Total Time
3	random	7	0.06231	0.00627	0.13482	0.27
	naive	9	0.06186	0.00038	0.17223	0.31767
	mean	9	0.06186	0.00043	0.17448	0.30433
	median	9	0.06186	0.00054	0.17590	0.311
7	random	8.6	0.02110	0.00650	0.36094	0.501
	naive	6	0.02181	0.00069	0.25252	0.391
	mean	6	0.02181	0.00075	0.25108	0.38767
	median	6	0.02181	0.00086	0.24742	0.381
10	random	7.8	0.01433	0.00599	0.45589	0.594
	naive	13	0.01308	0.00093	0.76347	0.89433
	mean	13	0.01308	0.00101	0.75557	0.90433
	median	13	0.01308	0.00112	0.75666	0.89767

As is visible from the results summarized in *Table 5.2*, random centroid initialization outperforms all sharding initialization algorithms for all values of k, as far as number of iterations and algorithm runtimes are concerned, for the Dessewffy dataset. However, and rather unexpectedly, all forms of sharding algorithm result in a superior mean SSE metric value than random initialization for 2 of the 3 values of k (3 and 10).

Jelonek

The Jelonek dataset is another artificial dataset created by the researcher. Jelonek is uniformly randomized in nature, similar to Dessewffy; however, this dataset consists of 2 attributes and 1000 instances, providing greater complexity by way of additional instances. This dataset is equally visually interpretable by the human eye as is Dessewffy, albeit much more densely distributed.

Figure 5.4 visualizes the Jelonek dataset.

Just as *Figure 5.2* demonstrated the difference between the centroid placement of a single occurrence of random initialization and naive sharding for the Dessewffy dataset, $k = 3$, *Figure 5.5* demonstrates the same for the Jelonek dataset.

Figure 5.6 demonstrates random versus naive sharding centroid initialization for $k = 7$.

Experiments for Jelonek included executing the testing regimen outlined above for k values of 3, 7, and 10. The results of these executions are outlined in

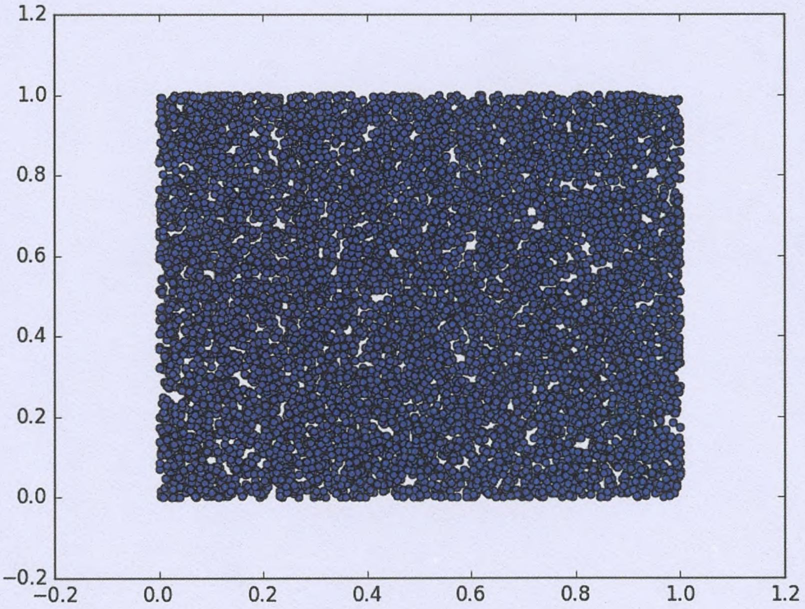


Figure 5.4: Jelonek dataset.

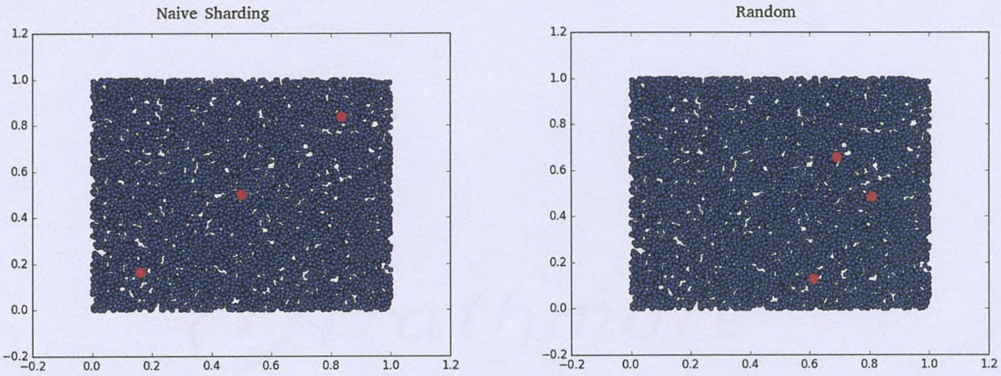


Figure 5.5: Jelonek naive sharding versus random initialization, $k = 3$.

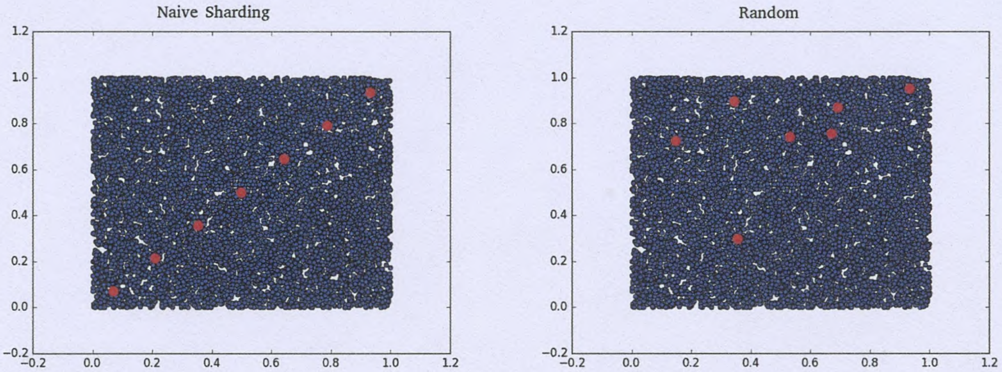


Figure 5.6: Jelonek naive sharding versus random initialization, $k = 7$.

Table 5.3: Jelonek results.

k	Init Type	Iters	Mean SSE	Init Time	Clust Time	Total Time
3	random	30.6	0.06686	0.58116	57.29435	58.03
	naive	15	0.06712	0.00311	28.1926	28.34767
	mean	15	0.06712	0.00322	28.27809	28.441
	median	15	0.06712	0.0035	28.1473	28.30767
7	random	29.7	0.0256	0.57949	120.27616	121.005
	naive	26	0.02584	0.00332	105.63985	105.791
	mean	26	0.02584	0.00338	105.66742	105.801
	median	26	0.02584	0.00381	105.57737	105.76433
10	random	48.3	0.01699	0.58077	274.7395	275.47
	naive	42	0.01701	0.00353	238.69629	238.84767
	mean	42	0.01701	0.00374	238.32835	238.49433
	median	42	0.01701	0.00406	237.40957	237.56767

Table 5.3.

As is visible in *Table 5.3*, the sharding family of centroid initialization algorithms outperformed random initialization in terms of number of iterations and all timing metrics (notice the dramatic difference in centroid initialization time for even a relatively small number of instances), for all values of k. Random initialization did have lower mean SSE scores for all values of k; however, these differences were extremely insiginifcant. For example, the mean SSE difference between all sharding algorithms and random initialization for $k = 3$ was 0.00026, the largest such difference exhibited of any value of k for the dataset.

5.2.2 Datasets of Known Clusters from the Literature

This section includes details and experiment results of well-known and -used datasets from the literature, of which the natural number of data clusters were known.

Ruspini

The Ruspini dataset[2] [18], similar to Dessewffy and Jelonek, is a 2-dimensional artificial dataset of which there are 75 instances. Unlike the others, however, Ruspini is not uniformly distributed, and consists of 4 natural clusters. Notably, the dataset does not include any classes, or distinguishing characteristics

---

[2] https://vincentarelbundock.github.io/Rdatasets/datasets.html

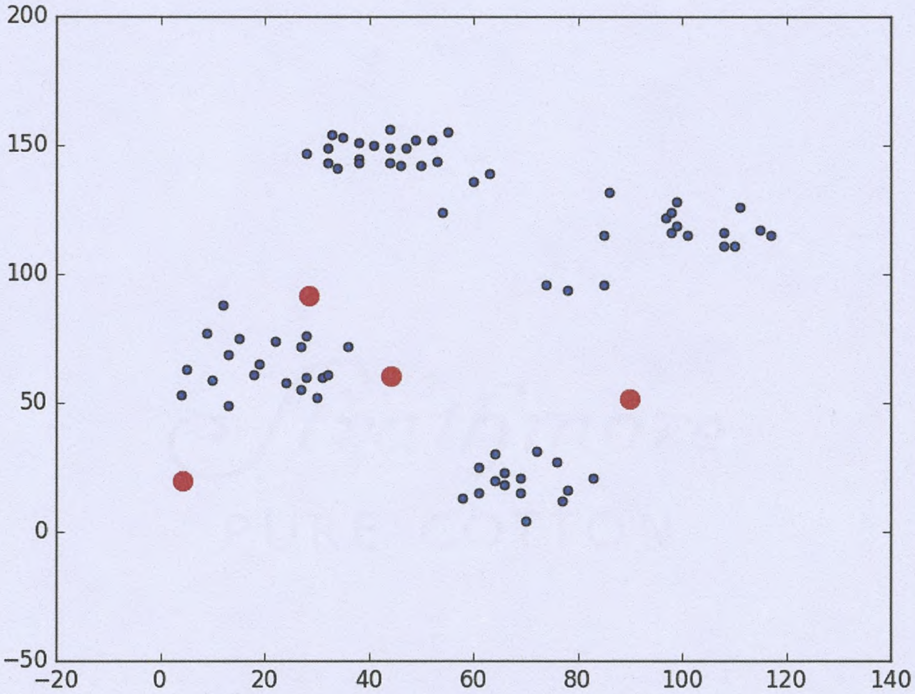


Figure 5.7: Ruspini random centroid initialization, $k = 4$.

of cluster membership, beyond the 2 data axes themselves. Data preparation for Ruspini included the removal of a header row, as well as a single column which consisted solely of an instance index.

Figure 5.7 visualizes the Ruspini dataset with a single occurrence of randomly-initialized centroids, $k = 4$. For comparison, *Figure 5.8* visualizes the Ruspini dataset with naive sharding centroid initialization, which produces the same centroids upon every execution, also with $k = 4$. *Figure 5.9* and *Figure 5.10* demonstrate the Ruspini dataset with 4 initial centroids generated via mean and median sharding algorithms, respectively. As is visible from comparison of

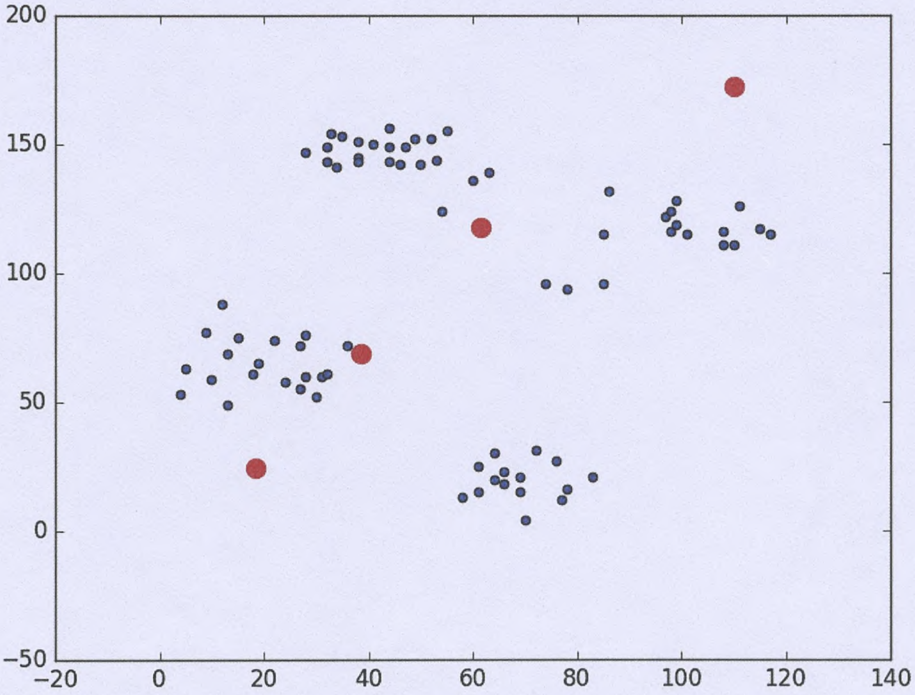


Figure 5.8: Ruspini naive sharding centroid initialization, $k = 4$.

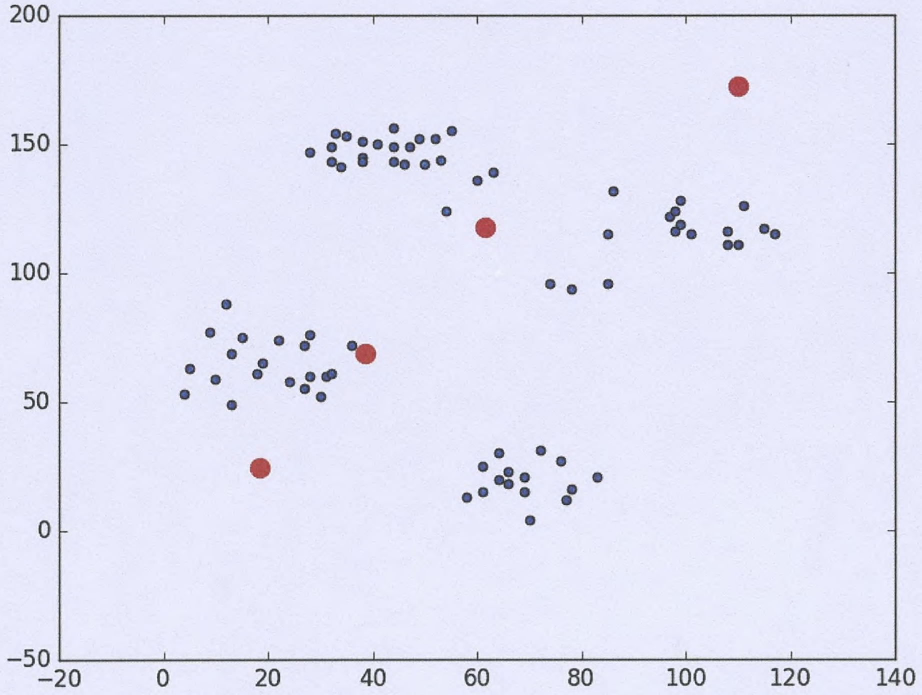


Figure 5.9: Ruspini mean sharding centroid initialization, $k = 4$.

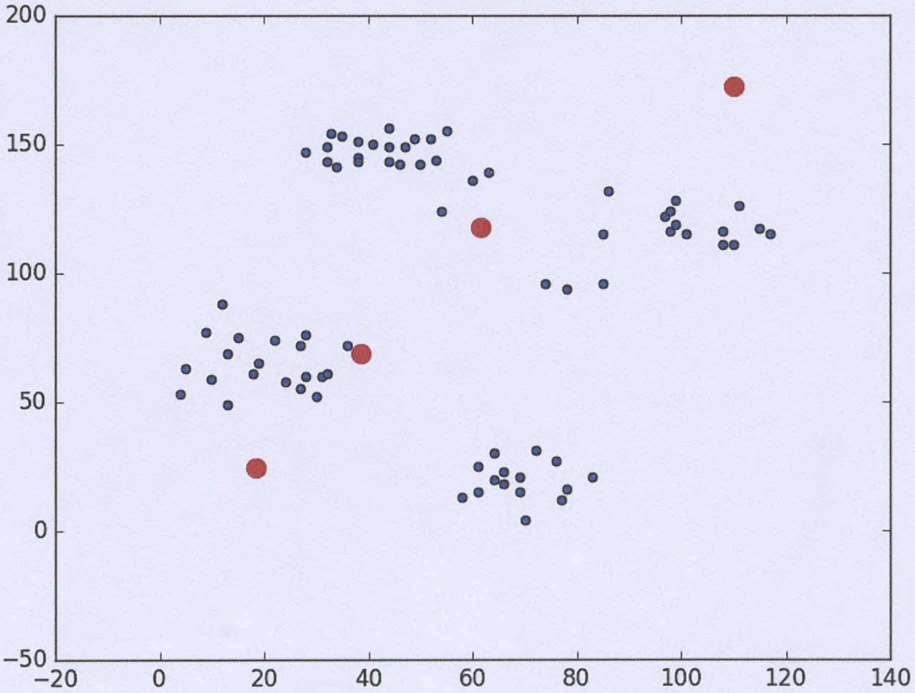


Figure 5.10: Ruspini median sharding centroid initialization, $k = 4$.

Table 5.4: Ruspini results.

Init Type	Iters	Mean SSE	Init Time	Clust Time	Total Time
random	3.2	271.425131	0.00465	0.06134	0.208
naive	4	171.74735	0.00049	0.07650	0.22433
mean	4	171.74735	0.00053	0.07619	0.20767
median	4	171.74735	0.00068	0.07713	0.22433

a dataset of this number and distribution of instances, these plots are virtually indistinguishable.

Experiments on Ruspini consisted of the regimen outlined in a previous section, solely for the k value of 4, the number of known clusters of the dataset. The results are outlined in *Table 5.4*.

Visible from the results in *Table 5.4*, random centroid initialization outperforms all sharding initialization algorithms in terms of number of clustering iterations and all time-related metrics, though the mean number of iterations are relatively similar, and the reduced execution time of the sharding algorithms keeps their collective overall execution times reasonably close to the overall random centroid initialization executions, despite the latter's lower number of clustering iterations required for convergence. However, and rather unexpectedly, the mean SSE metric of the sharding family of centroid initialization algorithms is considerably lower than for the random centroid initialization technique.

Iris

The Iris dataset[3] [19] is a very well-known natural dataset from the literature, and is the first dataset outlined in this research that is not artificially created. The Iris dataset contains data observations from 3 different types of irises, made up of 4 attributes (plus 1 class attribute) and 150 instances; the instances are of 3 classes, with an equal number of instances in each of these 3 classes.

Data preparation included remove the header row, as well as duplicating the dataset, removing the class column from one copy, and leaving it in for the other. The classes were converted from nominal to numeric values.

Experiments on Iris consisted of the regimen outlined in a previous section, solely for the k value of 3, the number of known clusters of the dataset. The experiment was executed for both dataset copies - with and without the class attribute - in order to observe the effect of the noise associated with clustering with attributes of immeasurable distance. The results for the Iris dataset without class attribute and with class attribute are outlined in *Table 5.5* and *Table 5.6*, respectively.

Meaningfully visualizing data to the human eye beyond 2 or 3 dimensions obviously becomes more difficult, given the lack of Euclidean plane or the human-friendly 3-dimensional space; however, the Iris dataset is presented in *Figure 5.11* using a parallel coordinate graph. This allows for some basic understanding of the data held in the Iris dataset.

---

[3] https://archive.ics.uci.edu/ml/datasets/Iris

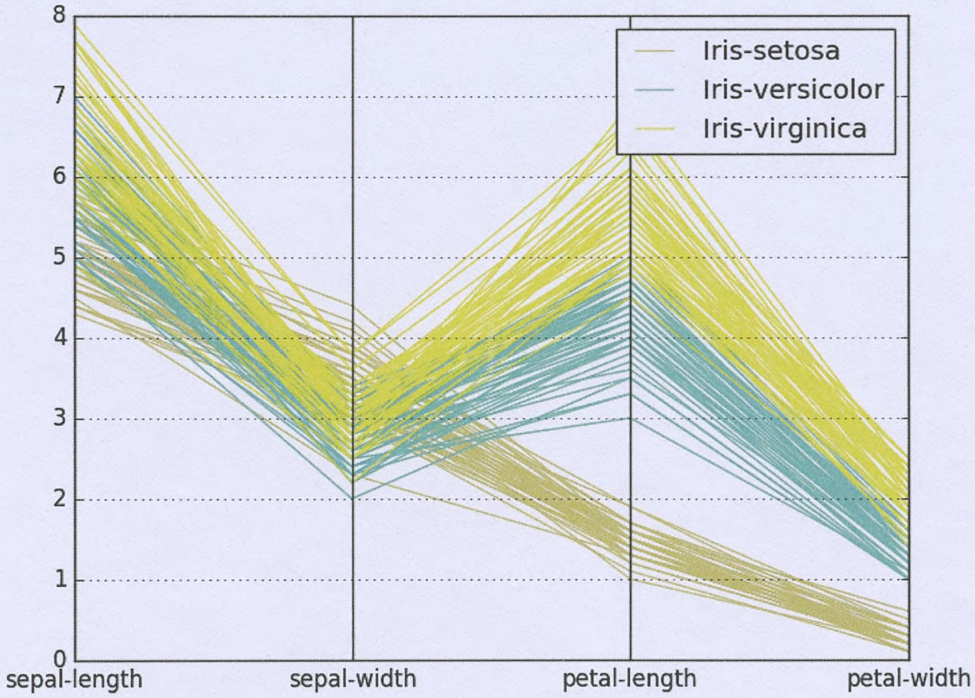


Figure 5.11: Iris parallel coordinates visualization.

Table 5.5: Iris results (without class).

Init Type	Iters	Mean SSE	Init Time	Clust Time	Total Time
random	7.5	0.52629	0.01771	0.21566	0.363
naive	3	0.52630	0.00043	0.08680	0.21433
mean	3	0.52630	0.00044	0.08676	0.22433
median	3	0.52630	0.00056	0.08663	0.22767

Table 5.6: Iris results (with class).

Init Type	Iters	Mean SSE	Init Time	Clust Time	Total Time
random	9	0.63744	0.02234	0.25698	0.412
naive	5	0.58207	0.00041	0.14394	0.27433
mean	5	0.58207	0.00045	0.14223	0.27767
median	5	0.58207	0.00054	0.14249	0.291

The Iris dataset results summarized in *Table 5.5* and *Table 5.6*, both with and without the class attribute, show that the sharding family of centroid initialization algorithms outperform random initialization in terms of all time-related metrics and number of iterations. Unexpectedly, sharding centroid initialization also results in a lower mean SSE value than does random initialization with the class attribute included; without the class attribute the mean SSE is nearly identical.

Wine

The Wine dataset[4] [20] is composed of chemical analysis data of wines grown in Italy. The dataset is made up of 13 attributes and 178 instances, which form 3 natural classes.

Data preparation of the Wine dataset included duplicating the dataset, and removing the class attribute from one of the copies.

Experiments on Wine consisted of the regimen outlined in a previous section, solely for the k value of 3, the number of known clusters of the dataset. The experiment was executed for both dataset copies - with and without the class attribute - in order to observe the effect of the noise associated with clustering with attributes of immeasurable distance. The results for the Wine dataset without class attribute and with class attribute are outlined in *Table 5.7* and *Table 5.8*, respectively.

As is evident from *Table 5.11*, the sharding family of centroid initialization

---

[4] https://archive.ics.uci.edu/ml/datasets/Wine

Table 5.7: Wine results (without class).

Init Type	Iters	Mean SSE	Init Time	Clust Time	Total Time
random	8.9	13909.19070	0.06757	0.29916	0.501
naive	5	13318.48139	0.00049	0.16849	0.30767
mean	5	13318.48139	0.00051	0.16781	0.28767
median	5	13318.48139	0.00068	0.16788	0.30767

Table 5.8: Wine results (with class).

Init Type	Iters	Mean SSE	Init Time	Clust Time	Total Time
random	11.8	14350.16163	0.07271	0.40031	0.609
naive	5	13318.77699	0.00049	0.17093	0.30433
mean	5	13318.77699	0.00054	0.16909	0.30433
median	5	13318.77699	0.0007	0.17014	0.301

algorithms outperformed random initialization in all time-related metrics, in number of clustering iterations required for convergence, as well as mean SSE, with the exception of reporting an identical mean SSE for one particular value of k ($k = 3$).

Seeds

The Seeds dataset[5] [21] consists of observations related to 3 different types of wheat. The dataset contains 210 instances and is made up of 7 attributes, plus 1 class attribute.

Data preparation tasks included duplicating the dataset, and removing the class attribute from one of the copies.

Experiments on the Seeds dataset consisted of the regimen outlined in a previous section, solely for the k value of 3, the number of known clusters of the

---

[5] https://archive.ics.uci.edu/ml/datasets/seeds

Table 5.9: Seeds results (without class).

Init Type	Iters	Mean SSE	Init Time	Clust Time	Total Time
random	7.9	2.80024	0.04317	0.31535	0.501
naive	4	2.80437	0.00047	0.1597	0.27767
mean	4	2.80437	0.00048	0.16028	0.291
median	4	2.80437	0.00059	0.15888	0.291

Table 5.10: Seeds results (with class).

Init Type	Iters	Mean SSE	Init Time	Clust Time	Total Time
random	8.2	2.97971	0.04910	0.32823	0.513
naive	5	2.97962	0.00046	0.20104	0.321
mean	5	2.97962	0.00050	0.20019	0.33433
median	5	2.97962	0.00059	0.20056	0.341

dataset. The experiment was executed for both dataset copies - with and without the class attribute - in order to observe the effect of the noise associated with clustering with attributes of immeasurable distance. The results for the Seeds dataset without class attribute and with class attribute are outlined in *Table 5.9* and *Table 5.10*, respectively.

As is visible from *Table 5.9* and *Table 5.10*, the sharding centroid initialization algorithms all outperformed random centroid initialization in all metrics, with the single exception being mean SSE in the experiments without the class attribute, in which random initialization reported a marginally lower 2.80024, as opposed to 2.80437.

5.2.3 Datasets of Unknown Clusters from Literature

This section includes details and experiment results of well-known and -used datasets from the literature, of which the natural number of data clusters were unknown.

3D Road Network

The 3D Road Network dataset[6] [22] was originally constructed by adding elevation information to a 2-dimensional road network in North Jutland, Denmark, and was subsequently used for benchmarking a number of fuel and CO2 estimation algorithms. It is of significantly larger size than the previously encountered datasets, consisting of 434874 instances of 4 attributes each. There are no known natural clusters for this dataset.

Data preparation for the 3D Road Network dataset was not required; the CSV file came ready for use.

Experiments for the 3D Road Network dataset included executing the testing regimen outlined above for k values of 3, 7, and 10. The results of these executions are outlined in *Table 5.11*.

As is visible from the results outlined in *Table 5.11*, the sharding centroid initialization algorithms all outperformed random centroid initialization in all metrics, with the single exception being mean SSE for $k = 3$, which was reported as identical to the random mean SSE. Of note, the centroid initialization times for a dataset of this size have been reduced by nearly 25000%; paired with lower numbers of iteration required for convergence and high-performing mean SSE values, the sharding family of centroid initialization algorithms are beginning to solidify as a viable alternative to traditional random centroid

---

[6] https://archive.ics.uci.edu/ml/datasets/3D+Road+Network+(North+Jutland,+Denmark)

Table 5.11: 3D Road Network results.

k	Init Type	Iters	Mean SSE	Init Time	Clust Time	Total Time
3	random	9.1	1.1965e+14	49.5408	726.1387	777.367
	naive	7	1.1965e+14	0.20484	564.12472	566.02767
	mean	7	1.1965e+14	0.20591	563.17698	565.031
	median	7	1.1965e+14	0.24226	563.31783	565.251
7	random	19.1	3.3969e+13	49.34861	3283.96743	3334.705
	naive	14	3.1036e+13	0.20907	2422.48985	2424.34433
	mean	14	3.1036e+13	0.20739	2408.66744	2410.55767
	median	14	3.1036e+13	0.24452	2414.75816	2416.69433
10	random	25.6	1.4968e+13	49.48046	6189.17081	6239.338
	naive	13	1.4713e+13	0.20754	3158.00158	3159.90767
	mean	13	1.4713e+13	0.20674	3137.79286	3139.41433
	median	13	1.4713e+13	0.24389	3159.00651	3160.96433

initialization.

Power Consumption

The Individual Household Electric Power Consumption, or simply Power Consumption, dataset[7] contains electric power consumption measurements in a single household over the course of approximately 4 years, with a one-minute sampling rate. The dataset consists of 9 attributes and 2049280 instances, making it the largest dataset used in this research.

Data preparation for the Power Consumption dataset required removing a number of instances (25979) which contained empty values, which the k-means clustering algorithm implementation is not designed to appropriately handle.

---

[7] https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption

Table 5.12: Power consumption results.

Init Type	Iters	Mean SSE	Init Time	Clust Time	Total Time
random	11.3	103.14419	412.00743	4286.46679	4707.652
naive	18	59.92447	1.33582	6843.3214	6853.99767
mean	18	59.92447	1.3415	6883.48539	6893.17433
median	18	59.92447	1.436	6874.63326	6884.43433

Experiments for the Power Consumption dataset originally included executing the testing regimen outlined above for k values of 3, 7, and 10; however, execution times for the k-means clustering algorithm on this dataset for any values of k greater than 3 proved prohibitive. This research decided on gathering the results for a value of $k = 3$, which are summarized in *Table 5.12*.

As evidenced by *Table 5.12*, random centroid initialization outperformed the sharding family of centroid initialization algorithms in terms of the number of clustering iterations required for convergence, as well as all time-related metrics, except for centroid initialization time. However, all of the sharding initialization methods reported mean SSE values over 40% lower than those reported by random initialization. Given the savings in execution time related to the centroid initialization times, along with the notable reduction in the mean SSE, the sharding centroid initialization algorithms do appear to have demonstrated some value on this much larger dataset.

Table 5.13 and *Figure 5.14* summarize select results of datasets of known clusters and dataset of unknown clusters, respectively. In each of these tables, the best-performing metrics are highlighted in red.

Strathmore
PURE COTTON

Table 5.13: Summarized results of datasets of known clusters.

Dataset	Init Type	Iters	Mean SSE
Ruspini	Random	3.2	271.425
	Naive	4	171.747
Iris (no class)	Random	7.5	0.52629
	Naive	3	0.52630
Iris (without class)	Random	9	0.63744
	Naive	5	0.58207
Wine (no class)	Random	8.9	13909.19070
	Naive	5	13318.48139
Wine (without class)	Random	11.8	14350.16163
	Naive	5	13318.77699
Seeds (no class)	Random	7.9	2.80024
	Naive	4	2.80437
Seeds (without class)	Random	8.2	2.97971
	Naive	5	2.97962

Strathmore
PURE COTTON

Table 5.14: Summarized results of datasets of unknown clusters.

Dataset	Init Type	Iters	Mean SSE
Dessewffy ($k = 3$)	Random	7	0.06231
	Naive	9	0.06186
Dessewffy ($k = 7$)	Random	8.6	0.02110
	Naive	6	0.02181
Dessewffy ($k = 10$)	Random	7.8	0.01433
	Naive	13	0.01308
Jelonek ($k = 3$)	Random	30.6	0.06686
	Naive	15	0.06712
Jelonek ($k = 7$)	Random	29.7	0.0256
	Naive	26	0.02584
Jelonek ($k = 10$)	Random	48.3	0.01699
	Naive	42	0.01701
3D Road Network ($k = 3$)	Random	9.1	1.1965e+14
	Naive	7	1.1965e+14
3D Road Network ($k = 7$)	Random	19.1	3.3969e+13
	Naive	14	3.1036e+13
3D Road Network ($k = 10$)	Random	25.6	1.4968e+13
	Naive	13	1.4713e+13
Power Consumption ($k = 3$)	Random	11.3	103.14419
	Naive	18	59.92447

Chapter 6

Conclusions

This chapter will summarize and analyze the results of the experiments run in the previous chapter, discuss the generalizability of the results, and draw warranted conclusions for the research. Afterward, future possible research will be suggested.

6.1 Summary of Research

The first observation gleaned from the results of last chapter's experiments is that the various members of the sharding family of centroid initialization algorithms - naive, mean, and median - performed almost identically throughout all tests on all datasets, with but a few minor exceptions. This observation is not necessarily generalizable based on the number of experiments; however, moving forward through analysis, the focus will be on the differences between random centroid initialization and a representative method of the sharding family - naive sharding centroid initialization.

As outlined in the previous section, the sharding family of centroid initialization algorithms did outperform random initialization more times than not in terms of number of iterations required for convergence and overall time-related metrics. The sharding family of centroid initialization algorithms also generally reported SSE values relatively similar to those reported by random centroid initialization. Rather unexpectedly, on occasion where sharding was outperformed by randomization in terms of execution time, it managed to report more favorable SSE values than did randomization.

In all experiments, rather unsurprisingly, the sharding initialization algorithms consistently outperformed randomization in terms of time required for centroid initialization, by very wide margins. It is this fact in terms of potential trade-off with competing factors that weighs heavily in favor of sharding initialization being viewed as a useful tool in general for k-means clustering.

For datasets with known numbers of clusters, sharding performed particularly and consistently well. *Figure 6.1* demonstrates the difference in iterations between random centroid initialization and naive sharding centroid initialization methods for the 4 datasets of known number of clusters. For 3 of these 4 datasets, naive sharding outperformed random initialization in terms of number of clustering iterations required for convergence, and subsequently all time-related metrics; the fourth set, Ruspini, reported mean numbers of iterations close to one another. Unexpectedly, for the Ruspini dataset, though

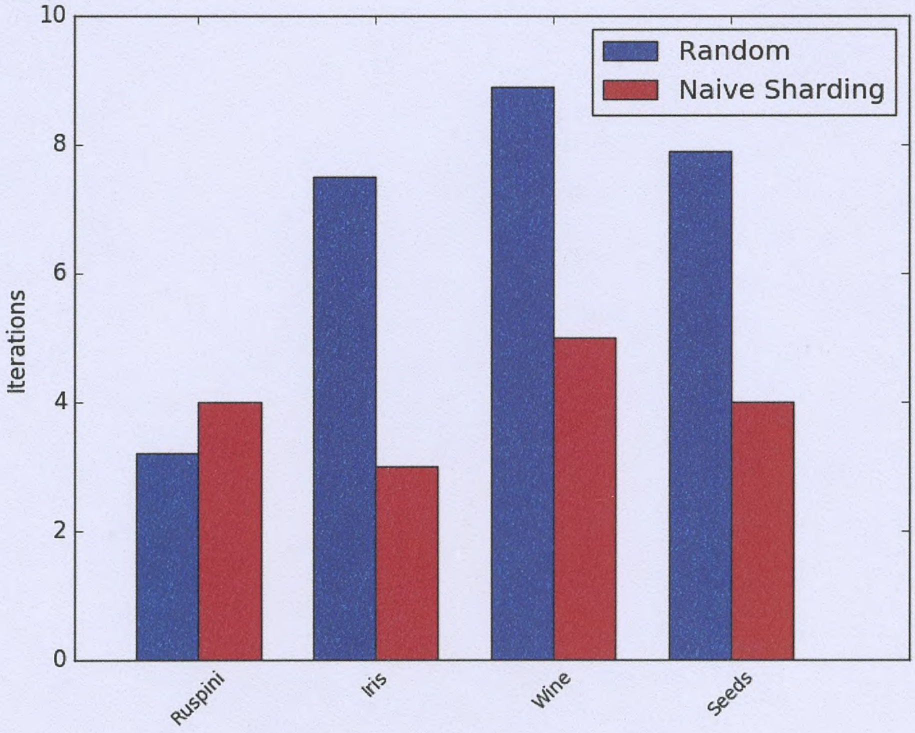


Figure 6.1: Number of iterations of datasets with known number of clusters.

naive sharding reported a slightly higher number of mean iterations (4 versus 3.2), its reported SSE values were lower, and thus more desirable, by a value of approximately 100, a more than 60% improvement; for all other of the known cluster number datasets, mean SSEs were equal, somewhat better, or very competitive by several thousandths of a decimal place.

The noise associated with the addition of class attributes were of no consequence; for the Iris, Wine, and Seeds datasets, sharding outperformed random initialization consistently for both dataset configurations, as demonstrated in Figure 6.2.

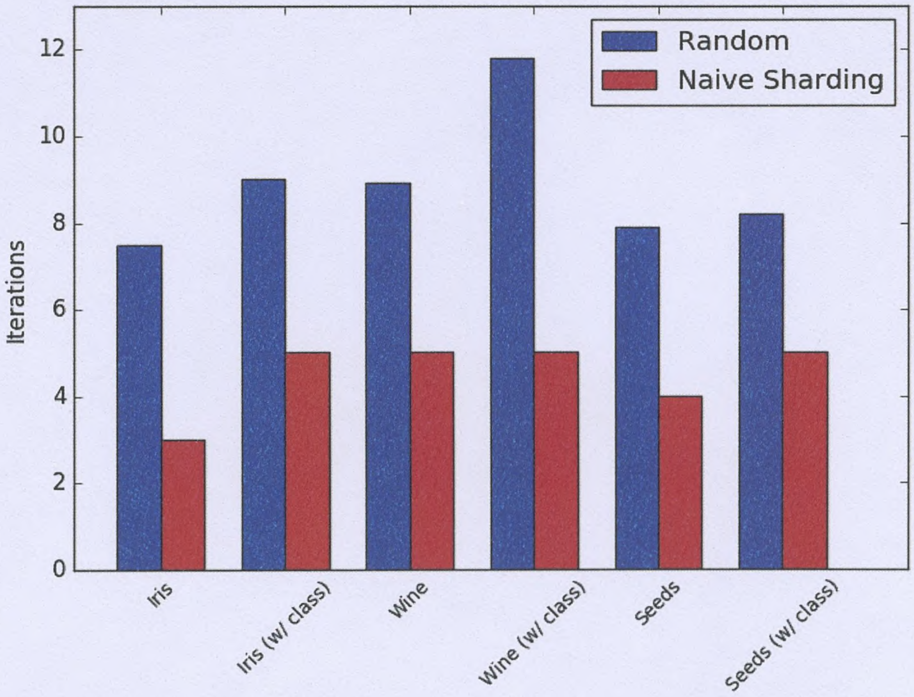


Figure 6.2: Number of iterations of datasets with class attributes.

Strathmore
PURE COTTON

In 2 of the 3 datasets of unknown number of clusters which employed multiple values of k for experimentation, sharding roundly outperformed randomization in the largest of the datasets, as well as the densest of the datasets, in number of required iterations and time-related metrics, and had either a very competitive or more desirable mean SSE.

The sharding initialization algorithms were outperformed in terms of number of iterations on the Dewsseffy dataset; however, 2 of these k values led to more favorable mean SSE values for the sharding algorithms. The sharding family was also outperformed in time-related metrics on the Power Consumption dataset, the largest dataset in the research; however, the sharding algorithms did report much better mean SSE values for this same dataset, suggesting that, even when the sharding algorithms are beaten at their own game of speed, the final clustering results are often likely as useful as randomization, if not significantly more so.

6.2 Future Work

Future research into the viability of this family of centroid initialization algorithms would likely be well-advised to focus on diversifying the test datasets, and performing an increased volume of algorithm testing. While limited by time to test the practical implementation, future work extending these experiments would be able to provide insight as to whether the results shared in the previous chapter could be generalized, and to what degree.

While this research cannot make such insinuations, nor did it intend to, being exploratory in nature, further research could help determine whether data density, complexity, or distribution consistently play a role in the success of the sharding family of centroid initialization algorithms.

While generalizing the results found herein would be irresponsible, based on the small sample size, the promising findings do suggest that further research is warranted, and that the sharding family of centroid initialization algorithms for the k-means clustering algorithm may, in fact, be another useful tool in the machine learning practitioner's toolkit when it comes to clustering analysis.

Appendix A

Computer Specifications

All research experiments were executed on a single computer of the specifications outlined in *Table A.1*.

Note that specifications were confirmed via the Linux *lshw*[1] command, which is used to extract detailed hardware information from a machine.

---

[1] http://linux.die.net/man/1/lshw

Table A.1: Experiment computer hardware.

Hardware	Model
Motherboard	Gigabyte H55M-UD2H
Processor	Intel(R) Core(TM) i5 CPU 760 @ 2.80GHz
L1 cache	64 KB
L2 cache	8 MB
System memory	16 GB (2 x 8 GB DIMM 1333 MHz (0.8 ns))
Display	Nvidia GeForce GTX 970
Audio	Nvidia GM204
Network	Realtek 8111/8168/8411 PCI Express Gigabit
Storage	640 GB Western Digital WD6402AAEX-0
Optical	Lite-On iHAS124

Appendix B

Artificial Dataset Names

I am a proponent of using Wikipedia randomization for naming almost anything. Not only does it remove the issue of having to come up with a name, or otherwise make some arbitrary choice, it allows for recreational learning at the same time. Hence, this was the method used for naming the artificially constructed datasets.

The first 2 random articles that Wikipedia returned during the naming process were used. Here is some further information for those interested in the dataset namesakes.

Dessewffy

Count Aurél Dessewffy de Csernek et Tarkeő[1] was a Hungarian politician and one-time Speaker of the House of Magnates, the National Assembly of

---

[1] https://en.wikipedia.org/wiki/Aurel_Dessewffy_(1846-1928)

Hungary's upper chamber.

Jelonek

Jelonek[2] is a village in the Gmina Niechanowo district of Greater Poland Voivodeship, located in west-central Poland.

---

[2] https://en.wikipedia.org/wiki/Jelonek,_Gniezno_County

Bibliography

[1] Mitchell, T.M. *Machine learning.* New Delhi, India: McGraw-Hill, 1997. ISBN: 9780070428072.

[2] Fayyad, U., Piatetsky-Shapiro, G. & Smyth, P. "From data mining to knowledge discovery in databases". In: *Advances in Knowledge Discovery and Data Mining.* Ed. by Fayyad, U., Piatetsky-Shapiro, G., Smyth, P. & Uthurusamy, R. Cambridge, MA: MIT Press, 1996, pp. 1–36. URL: `http://www.csd.uwo.ca/faculty/ling/cs435/fayyad.pdf`.

[3] Jiawei, H., Kamber, M. & Pei, J. *Data mining: Concepts and techniques, 3rd ed.* San Francisco, CA: Morgan Kaufmann Publishers, 2012. ISBN: 9780123814791.

[4] Bishop, C. *Pattern recognition and machine learning.* Singapore: Springer, 2006. ISBN: 9780387310732.

[5] Wu, X., Kumar, V., Quinlan, J.R., Ghosh, J., Qiang, Y., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Yu, P.S., Zhou, Z. Steinbach, M., Hand, D.J., & Steinberg, D. "Top 10 algorithms in data mining". In: *Knowledge Information Systems* 14 (2008), pp. 1–37. DOI: `10.1007/S10115-007-`

0114-2. URL: http://www.cs.uvm.edu/~icdm/algorithms/10Algorithms-08.pdf.

[6] Arthur, D. & Vassilvitskii, S. "k-means++: The advantages of careful seeding". In: (2006). URL: http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf.

[7] Chaturvedi, A., Green, P.E. & Carroll, J.D. "K-modes clustering". In: *Journal of classification* 18 (2001), pp. 35–55. DOI: 10.1007/s00357-001-0004-3. URL: https://www.researchgate.net/profile/Anil_Chaturvedi2/publication/226946703_K-modes_Clustering/links/54c99baf0cf298fd26264cac.pdf.

[8] Steinbach, M., Karypis, G. & Kumar, V. *A comparison of document clustering techniques*. Tech. rep. URL: http://cs.fit.edu/~pkc/classes/ml-internet/papers/steinbach00tr.pdf.

[9] Su, T. & Dy, J. "A deterministic method for initializing k-means clustering". In: (). URL: https://pdfs.semanticscholar.org/d853/79ef05fd9b02f71f947f1b58df474773767f.pdf.

[10] Gingles, C & Celebi, M.E. "Histogram-based method for effective initialization of the k-means clustering algorithm". In: *Proceedings of the twenty-seventh international florida artificial intelligence research society conference*. 2014, pp. 333–338. URL: http://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS14/paper/view/7782/7864.

[11] Oliphant, T. "Python for scientific computing". In: *Computing in science and engineering* May/June (2007), pp. 10–20. DOI: 10.1109/MCSE.2007.58. URL: https://www.researchgate.net/profile/Travis_Oliphant

/publication/3422935_Python_for_Scientific_Computing/links/548f0b680cf225bf66a7f9c3.pdf.

[12] TIOBE. *TIOBE Index for February 2016*. 2016. URL: http://www.tiobe.com/index.php/tiobe_index (visited on 02/05/2016).

[13] Hilley, D. *Python: Executable pseudocode*. Georgia Institute of Technology. University lecture. DOI: 10.1.1.211.7674. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.211.7674&rep=rep1&type=pdf.

[14] Scipy. *Numpy and Scipy documentation*. 2016. URL: http://docs.scipy.org/doc/ (visited on 03/10/2016).

[15] Harrington, P. *Machine learning in action*. Shelter Island, NY: Manning Publications, 2012. ISBN: 9781617290183.

[16] Python. *Our documentation*. 2016. URL: https://www.python.org/doc/ (visited on 02/22/2016).

[17] Die.net. *time(1) - Linux man page*. URL: http://linux.die.net/man/1/time (visited on 03/28/2016).

[18] Ruspini, E.H. "Numerical methods for fuzzy clustering". In: *Information sciences* 2 (3 1970), pp. 319–350. DOI: 10.1016/S0020-0255(70)80056-1.

[19] Fisher, R.A. "The use of multiple measurements in taxonomic problems". In: *Annual Eugenics* 7 (1936), pp. 179–188.

[20] Aeberhard, S., Coomans, D. & Vel, D. "The performance of statistical pattern recognition methods in high dimensional settings". In: ().

[21] Charytanowicz, M., Niewczas, J., Kulczycki, P., Kowalski, P.A., Lukasik, S. & Zak, S. "A complete gradient clustering algorithm for features analysis of x-ray images". In: *Information technologies in biomedicine.* Ed. by Pietka, E. & Kawa, J. Berlin-Heidelberg, Germany: Springer-Verlag, 2010, pp. 15–24.

[22] Kaul, M., Yang, B. & Jensen, C.S. "Building accurate 3D spatial networks to enable next generation intelligent transportation systems". In: *Proceedings of international conference on mobile data management.* Milan, Italy, 2013, pp. 137–146. DOI: `10.1109/MDM.2013.24`.

245 L5 A173 IFH Group
07/08/16 J 31180